Modellbildung und computergestütztes Modellieren in frühen Phasen des architektonischen Entwurfs

Dissertation

zur Erlangung des akademischen Grades

Doktor-Ingenieur

an der Fakultät Bauingenieurwesen
der
Bauhaus-Universität Weimar

vorgelegt von

Dipl.-Ing. Frank Steinmann

geb. am 10 Juni 1962

in Sömmerda

Tag der Disputation: 23. Juli 1997

Gutachter:
1. Prof. R. Hübler
2. Prof. N. Kohler
3. Prof. D. Donath

DANKSAGUNG

Mein Dank gilt zuerst Herrn Prof. Dr. R. Hübler für seine fachliche Betreuung, seine Geduld und nicht zuletzt für seinen ständigen Ansporn. Während meiner Tätigkeit an dem von ihm geführten Lehrstuhl für „Informations- und Wissensverarbeitung“ hat er mir die Freiräume geschaffen, die eine Voraussetzung für das Schreiben dieser Arbeit waren.

Herrn Prof. Dr. D. Donath danke ich für inhaltlichen Anregungen und kritische Bewertung aus Sicht der Architektur.

Mein Dank gilt weiterhin den Herren Dipl.-Inf. P. Kolbe und Dipl.-Inf. R. Wehner, die wesentlich an der technischen Umsetzung beteiligt waren. In diesem Zusammenhang bedanke ich mich auch bei den Herren Dipl.-Inf. S. Braxein, Dipl.-Inf. C. Langenhan, Dipl.-Inf. B. Brettschneider sowie bei Frau Dipl.-Inf. S. Sklenar.

Herrn Dr. D. Ranglack dank ich für seine stetige Bereitschaft zur Diskussion, so daß die Umsetzbarkeit der dargestellten Ideen in der Industrie nicht vernachlässigt wurde.

Bedanken möchte ich mich auch bei den Kolleginnen, die mir bei der Korrektur des Manuskriptes behilflich waren.

Nicht zuletzt möchte ich mich bei meiner Familie bedanken. Sie gab mir Rückhalt und die Möglichkeit zu ungestörtem Arbeiten.

KURZREFERAT

Die Arbeit befaßt sich mit der Konzeption und exemplarischen Realisierung von CAD-Tools zur Unterstützung des konzeptuellen Entwurfs von Gebäuden. Die Spezifik des Bauwerksentwurfs macht eine Analyse des Arbeitsfelds ‘früher architektonischer Entwurf’ aus Sicht der Informationsverarbeitung nötig. Auf der Grundlage einer Untersuchung des Entwurfsprozesses aus pragmatischer und kognitiver Sicht wird ein generisches Prozeßmodell ‘Entwurf’ abgeleitet und mit Beispielen belegt.

Ausgehend von der Auffassung, daß Entwerfen der Prozeß des Vorausdenkens von Bauwerken in Modellen ist, wird die ‘Objektorientierte Modellierungstechnologie’ einer eingehenden Bewertung unterzogen. Ihre methodische Eignung zur modellhaften Abbildung von Bauwerken wird grundsätzlich nachgewiesen. Mit Blick auf die spezifischen Gegebenheiten der frühen Entwurfsphasen, wie Unschärfe und nicht formalisiert darstellbarer Informationen, werden einige Erweiterungen vorgeschlagen und experimentiert.

Ziel der Arbeiten ist es, ein multipel und über lange Zeiträume auswertbares Bauwerksmodell zu schaffen, um Planungsprozesse über den gesamten Bauwerkslebenszyklus durchgängig zu unterstützen.

Grundlage des Entwerfens mittels der objektorientierten Methodik sind generalisierte Domänenmodelle. Es wird gezeigt, daß für eine Interpretation konkreter Bauwerksmodelle die generalisierten Domänenmodelle entweder zu normieren sind oder aber daß sie explizit verfügbar sein müssen. Die Objektorientierung bietet die Möglichkeit, sowohl konkrete als auch generalisierte Modelle in einem einheitlichen Metamodell zu beschreiben. Durch die explizite Verfügbarkeit der Domänenmodelle wird es weiterhin möglich, daß der Nutzer dieses Modell dynamisch anpaßt, erweitert und sich damit ein ‘privates’ CAD-Tool schaffen kann.
Da diese Tools über dynamischen Modellwelten arbeiten, wird von einer reduzierten Berechenbarkeit in solchen Modellen ausgegangen. Im Sinne des angestrebten Systemcharakters ‘Design Repository’ rückt die Erfassung, Strukturierung und Bewahrung von Bauwerksmodellen in den Vordergrund.

Die diskutierten Ansätze wurden prototypisch in Form der Tools FUNPLAN (funktionaler Entwurf), INFPLAN (Authoring und Retrieval informaler Modellerweiterungen) sowie NETGEN (Generieren von Graphgeometrien aus topologischen Relationen der funktionalen Spezifikation) realisiert und bewertet. Zur Integration der Komponenten wurde das CAAD-Systemkonzept PREPLAN entwickelt.

Die softwaretechnische Basis derartiger Tools bilden dynamische Modellverwaltungssysteme. Diese Systeme müssen neben den Modelldaten (Instanzobjekten) auch die Elemente des Domänenmodells (Klassenobjekte) speichern und bearbeiten können. Solche Basissysteme sollten eine weitgehende Analyse der Taxonomie sowie eine Reklassifikation von Modellinstanzen unterstützen. Mit den Systemen ALOS (AutoLisp-Erweiterung) und FLEXOB (C++ basiertes flexibles Objektsystem) werden zwei realisierte Varianten vorgestellt.

GLIEDERUNG

ANHANG

0. VORBEMERKUNG

Praktischer Anlaß der vorliegenden Forschungsarbeit war der Wunsch, zu Ausbildungszwecken an der Fakultät Architektur ein CAD-Tool zu schaffen, mit dem der funktionsorientierte Entwurf unterstützt wird. Bei der Diskussion der Frage „Was kommt vor der Form?“ fiel auf, daß gerade systematische Entwurfstechniken, wie eben der funktionale Entwurf, nicht durch CAAD-Werkzeuge unterstützt sind. Dies ist um so bedauerlicher, da die in dieser frühen Planungsphase erzeugten Beschreibungen des zukünftigen Gebäudes am Anfang einer Prozeßkette stehen, die von der Planung des Bauwerks über seine Nutzung bis hin zum Abriß reicht. Es zeigte sich, daß der systematischen Erfassung von Entwurfsgrundlagen in Form funktionaler Modelle große Bedeutung für eine durchgängige Computerstützung zukommt. Hier wird der Kern einer Modellstruktur geschaffen, die in differenzierter Weise das Gebäude über alle Lebensphasen beschreibt.

Bei den inhaltlichen Überlegungen rückten folgende Fragen ins Zentrum des Interesses:

- *Woran arbeitet der Architekt ?* (⇨ Aspekte des Entwerfens)
- *Wie arbeitet der Architekt ?* (⇨ Vorgehensmodelle des Entwerfens)
- *Womit arbeitet der Architekt ?* (⇨ Modellelemente des Entwurfs).

Mit Hilfe der funktionalen Spezifikation lassen sich im Zuge des Entwurfsprozesses immer wieder Entwurfsvarianten am geplanten Nutzungszweck messen. Im vorgestellten CAD-Systemkonzept PREPLAN spielt deshalb die Funktionsplanung eine zentrale Rolle. Da seitens der Architekten der Wunsch bestand, gebräuchliche Dokumentationsformen wie Skizzen, Texte aber auch gesprochene Sprache und Videosequenzen als Ausdrucksmittel im frühen Entwurf nutzen zu können, wurde das PREPLAN-Konzept um die Möglichkeit der Integration solcher Dokumente in das formale CAD-Modell erweitert.

Bei den Untersuchungen zur Realisierung des geplanten Tools wurde festgestellt, daß die Entwicklungs- und Implementierungstechnologie der ‘Objektorientierung’ direkt als Modellierungsparadigma für die Funktionsplanung geeignet ist. Es zeigte sich jedoch, daß sich fixierte, generalisierte Domänenmodelle, wie sie gegenwärtig die Grundlage eines Softwareentwicklungsprozesses bilden, für die zu unterstützende Planungsphase nur bedingt finden lassen. Unschärfe, Subjektivität und Dynamik sind hier inhärente Modellmerkmale, die es in entsprechenden CAD-Tools zu akzeptieren bzw. zu unterstützen gilt.

Im Zuge der Realisierung der angestrebten Entwurfsunterstützung war die Entwicklung von Basissoftware zur Modellverwaltung notwendig. Wegen ihrer allgemeinen Anwendbarkeit ist diese Software für weitere Entwicklungen interessant.

Den organisatorischen Rahmen dieser Arbeit bildet das Forschungsprojekt ‘Intelligente CAAD-Systeme’ innerhalb des DFG-Forschungsschwerpunktes ‘Objektorientierte Modellierung in Planung und Konstruktion’[1], das in einem Zeitraum von vier Jahren am Bereich ‘Informations- und Wissensverarbeitung’ der Bauhaus-Universität Weimar durchgeführt wurde.

Die Arbeit befindet sich am Schnittpunkt mehrerer Wissenschafts- und Technikgebiete, die einen jeweils eigenständigen Begiffsapparat entwickelt haben. Die Benutzung von Fachvokabeln ist jedoch häufig uneinheitlich, so daß eine Begriffsdefinition nötig war, um eine konsistente Begrifflichkeit zu gewährleisten. Es sei an dieser Stelle ausdrücklich auf das Glossar im Anhang der Arbeit verwiesen.

[1] Fördernummer: HU519/1-4

1. CAD – GRUNDLAGEN UND MOTIVATION

1.1 Erkenntnistheoretischer Rahmen der CAD-Systementwicklung und Nutzung

Der Gebrauch von Werkzeugen war ein Meilenstein auf dem Weg der Menschwerdung. Die zielgerichtete Herstellung und Nutzung von Dingen, die unsere natürlichen Möglichkeiten um neue Fähigkeiten erweitern, zeichnet den Menschen aus. Mit dem Beginn des Computerzeitalters zählen hierzu auch Fähigkeiten, die auf dem Gebiet geistiger Leistungen liegen. Kaum eine Technologie hat die Phantasie und Erwartungen der Menschen so bewegt wie die Computertechnologie. Die Annahme, daß in 'Elektronenhirnen' dem menschlichen Denken ähnliche Prozesse ablaufen oder doch ablaufen können, führte zur Erwartung, daß in kurzer Zeit mechanischen Werkzeugen analoge 'Denkzeuge' verfügbar wären, die Tätigkeiten automatisieren, zu deren Ausführung Intelligenz erforderlich ist. NEWELL und SIMON behaupteten 1958 [NEW58]

> „... zumindest in begrenzten Gebieten wissen wir heute nicht nur, wie man Computer programmieren muß, damit sie erfolgreich Probleme lösen können; wir wissen auch wie man sie programmieren muß, damit sie diese Fähigkeiten *erlernen*. Kurz gesagt, wir kennen nun die Elemente einer Theorie des heuristischen Problemlösens ... mit dieser Theorie können wir sowohl heuristische Prozesse im Menschen verstehen als sie auch auf Digitalcomputern simulieren. Intuition, Einsicht und Lernen sind nicht länger ausschließlich im Besitz des Menschen: Jedem Hochgeschwindigkeitscomputer können sie einprogrammiert werden"

Das Planen von Produkten ist ein herausragendes Beispiel für eine solche Intuition, Einsicht und Lernen erfordernde Tätigkeit. Das Ergebnis dieser Tätigkeit ist ein immaterieller Plan – das Finden dieses Plans und nicht vordergründig die Arbeiten zu seiner Dokumentation mittels eines geeigneten Mediums sind Ziel des Planungsprozesses.

Wie weit wir auf dem Weg hin zu den gewünschten 'Denkzeugen' gekommen sind, wird im weiteren kontrovers diskutiert werden müssen (siehe Kapitel 2). Daß gerade beim Planen die Praxis weit von ihnen entfernt ist und auch in der nahen Zukunft keine wirklich kognitiven oder kreativen Verstandesleistungen von symbolverarbeitenden Computern erbracht werden können, belegen VARELA [VAR90], DREYFUS/DREYFUS [DRE91], BECKER [BEC92] u.a.

Realisierbar im Bereich des technischen Entwerfens scheint der Wunsch langweilige, ermüdende und vermeintlich anspruchslose Tätigkeiten dem Computer zu übertragen, um so die Effizienz und Qualität des menschlichen Schaffensprozeß zu steigern. Der Mensch gewinnt Zeit für anspruchsvolle, ihn fordernde Aufgaben.

Darüber hinaus besteht aber auch der Wunsch, der Computer möge (Entwurfs-) Entscheidungen treffen und so den Entscheidungsträger von Verantwortung befreien. Das Verdikt des Rechner scheint unabhängig, objektiv, reproduzierbar und unangreifbar. Er kennt keine Konzentrationschwächen oder 'private' Vorlieben und macht (folgt man den Versprechungen seiner Protagonisten) Expertise beliebig verfügbar. Schon in den Denkerschulen des alten Griechenland wurde nach Verfahren gesucht, solche objektiven Entscheidungsverfahren zu gewinnen. PLATON wollte schon 450 v.Chr. den wissenschaftlichen Disput formalisieren, um Subjektivität auszuschließen und formulierte, daß „sich alles Wissen in eindeutigen Definitionen formulieren lassen" .. müsse .. „die jeder (also auch Computer, Anm. d. Autors) anwenden kann".

Durch das Sammeln von Wissen und dessen Formalisierung in Form von Regeln (Logik) versprach der damals entwickelte Syllogismus objektive Schlüsse – wäre nur das Wissen objektiv, wäre durch die gegebene Verfahrensweise eine 'richtige' Entscheidung garantiert. Aus dieser Tradition heraus schien der zur Symbolverarbeitung fähige Computer das ideale Werkzeug, derartige Prozesse zu mechanisieren. Das Metapher des 'Elektronenhirns' unterstreicht diesen Anspruch.
Eine zentrale Annahme solchen Vorgehens ist die Existenz abgrenzbarer mentaler Repräsentationen eines beobachtbaren Phänomens sowie die Annahme, daß Verstandesleistungen durch Räsonieren, d.h. durch eine Art von 'Verrechnung' solcher Repräsentationen, erbracht werden. Eine weitere Grundannahme ist, daß sich sowohl diskrete mentale Repräsentationen als auch die 'Verrechnungsverfahren' externalisieren lassen. Diese Thesen werden von der Schule der Repräsentationalisten vertreten, die sich in der Tradition von PLATON, DESCARTES und KANT sehen[2]. Allerdings bleibt bei dieser Sicht auf das praktische Können des Menschen seine Intuition und sittliche Einsicht außerhalb der Betrachtung[3].
Diametral ist der Ansatz der Behavioristen, die eine diskrete Repräsentation verneinen[4]. Sie stellen den Begriff der 'Zweckorientiertheit' (Pragmatismus) in den Mittelpunkt. Es wird postuliert, daß jede bewußte oder unbewußte Handlung (und damit letztlich jegliche Lebensäußerung) eine Reaktion auf eine ganzheitliche Umweltsituation darstellt. Zweckorientiertheit entsteht einzig durch die Korrelation von Situation und Handlung. Das Aufdecken dieser Korrelation oder besser deren Wirkprinzipien durch Introspektive ist nicht möglich. Diese Theorie erklärt sehr gut intelligentes Verhalten in einem homogenen System von Lebensäußerungen, die von Stoffwechselreaktionen bis hin zu kreativem Verhalten reicht. Die Repräsentation von Wissen erfolgt holistisch, nicht in diskreten Symbolen. Die Theorie basiert letztlich auf den Experimenten von PAVLOW (1925) zu konditioniertem Verhalten. Der Konnektivismus[5] setzt diese Doktrin unter Nutzung der Hebbschen Lernregel[6] technisch in neuronalen Netzen um.

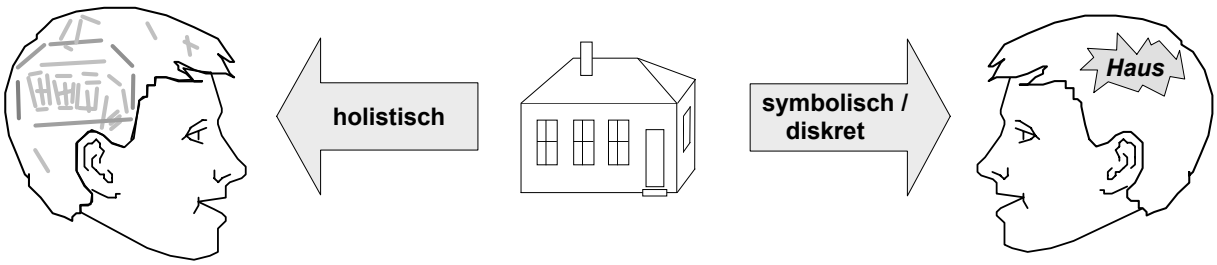


Abb. 1-1 Charakter mentaler Widerspieglungen

[2] KANT in seiner Kritik der reinen Vernunft: „Denken ist Erkenntnis durch Begriffe"
[3] Sittliche Einsicht bezeichnet ein moralisches Wissen, ist also durch den gesellschaftlichen Kontext geprägt. Intuition oder auch Instinkt ist zielgerichtetes und im allgemeinen erfolgreiches Handeln, das vom situativen Kontext unabhängig scheint.
[4] Eine eingehende Darstellung findet sich in bei GARDNER [GAR92]
[5] eine Einführung findet sich etwa bei DORFNER [DOR91]
[6] HEBB (englische Psychologe) formulierte 1949, daß wenn zwei Nervenzellen miteinander verbunden sind und beide Zellen erregt sind, eine Verstärkung der Verbindung erfolgt, d.h. die Korrelation der Erregungszustände stärker wird. Man sieht, daß Lerneffekt lokal ist und völlig ohne Kenntnis einer 'Außenwelt' stattfindet. Die Regel beschreibt beginnend mit dem legendären 'PERCEPTRON' den Grundmechanismus der meisten technischen Neuronalen Netze.

Bei allem Dissens, der unter Philosophen, Kognitionswissenschaftlern und KI-Forschern besteht[7], bleibt festzustellen, daß beide Ansätze bestimmte Aspekte menschlichen Handelns gut erklären. Daß Introspektive und symbolisches Schließen funktionieren, weiß jeder, der ein Brettspiel beherrscht. Andererseits können wir Radfahren und erkennbaren Gefahren ausweichen, völlig ohne bewußte 'Denkvorgänge'. Für beide Extreme gibt es folgerichtig auch Computeranwendungen, die dem jeweiligen Paradigma folgen.

Wie sich eine beabsichtigte Stützung von Entwurfsprozessen in das Spektrum der kognitiven Theorien einordnet, hängt in starkem Maße von den verfügbaren Hard- und Softwaretechnologien ab. Diese Technologien, die von einem rationalistisch/ repräsentationalistischen Ansatz ausgehen, bestimmen Funktionalität und Funktionsweise heutiger Softwaresystemen. In bestimmten Anwendungsdomänen wie dem frühen architektonischen Entwurf, arbeiten sie allerdings nur inakzeptabel. Auch die vorliegende Arbeit geht jedoch primär von einem Repräsentationalistischen Ansatz aus. Architektonisches Entwerfen ist zwar nicht Schachspielen, trotzdem haben sich komplexe symbolische Repräsentationen, d.h. Modelle der Artefakte als Arbeitsmittel bewährt. Sie sind die Basis jeglicher Kommunikation im Entwurfsprozeß. Die Arbeit folgt aber den weiteren zwei Prämissen des Rationalismus, nämlich der Externalisierbarkeit und der Verrechenbarkeit, nur bedingt. Das Räsonieren über Modellwelten ist nicht Gegenstand dieser Arbeit. Externalisierbarkeit muß in einem arbeitsteiligen Prozeß jedoch gegeben sein. Sie ist eine Grundvoraussetzung zum Modellaustausch und zur Modellbewahrung und damit Grundlage jeglicher programmierter Unterstützung im Rahmen der verfügbaren Softwaretechnologien. Im Gegensatz zum Rationalismus wird jedoch keine Objektivität der Modellwelten vorausgesetzt, d.h. Subjektivität wird bei der Interpretation der erstellten Symbolwelten anerkannt. Dies hat zur Folge, daß in diesen subjektiven Teilen von Modellwelten programmiertes Räsonieren nicht möglich sein wird. Das Anliegen der Arbeit ist also im philosophischen Sinne die Externalisierung mentaler Modellwelten als Basis von Kommunikation, wobei Informationsverlust durch private Interpretation akzeptiert wird. Dies ist *kein* Mangel der angestrebten Lösung, sondern im Charakter jeglicher Kommunikation begründet, wie bei MATURANA ([MAT87]) bzw. WINOGRAD ([WIN86]) gezeigt wird. Kommunikation ist immer die Wechselwirkung zweier Subjekte[8], mithin kann Objektivität nicht das Ziel sein, sondern bestenfalls Objektivierung.

1.2 Motivation der Stützung früher Phasen der Bauwerksplanung

Mit der Technischen Revolution im 19. Jahrhundert hat sich der Prozeß der Realisierung gesellschaftlicher Bedürfnisse entscheidend gewandelt. Bedingt durch eine Mechanisierung und Automatisierung der eigentlichen Herstellungsprozesse und der damit verbundenen Steigerung der Arbeitsproduktivität ist eine Aufwandsverschiebung von der Herstellung hin zur Planung zu verzeichnen. Mit Planung ist hierbei der Prozeß der Erstellung aller zur Produktion nötigen informationellen Gundlagen auf der Basis einer technischen Aufgabenstellung gemeint ([MOR87]). Diese Verschiebung führt zu einer Diskrepanz, die sich im starken Anstieg

[7] die Diskussion zwischen Anhänger beider 'Lager' wird vor allem in den USA sehr polemisch geführt, die bis hin zu persönlichen Angriffen reicht. In diesem Zusammenhang wurde der Begriff des 'Symbol Wars' geprägt. Letztlich zeigt die heftige Diskussion von welch immenser Bedeutung diese Frage ist.

[8] in [MAT87]: „..., daß es biologisch gesehen in der Kommunikation keine 'übertragenen Informationen' gibt. Es gibt Kommunikation jedesmal, wenn bei gegebener Strukturkopplung Verhaltenskoordination auftritt." und weiter „Das Röhrenmetapher gilt nicht. ... Jede Person sagt, was sie sagt, und hört was sie hört, gemäß ihrer eigenen Strukturdeterminiertheit; daß etwas gesagt wird, garantiert nicht, daß es auch gehört wird", d.h. das es zur Verhaltenskoordination führt (Anm. des Autors)

der Anforderungen an den Entwurfsprozeß einerseits und dem dürftigen Angebot technischer Unterstützung dieses Prozesses andererseits äußert (siehe Abb. 1-2).
Das angesprochene Problem ist hinlänglich bekannt, interessant ist jedoch, daß sich in gleicher Weise eine Diskrepanz im Planungsprozeß selbst wiederfinden läßt. So ist heute die (physische) Erstellung jeder Art medialer Präsentationen sowie die Ausführung komplexer oder langwieriger Berechnungen unterstützt oder wenigstens unterstützbar. Die Erstellung der Konzeption, d.h. des initialen Modellkerns des zu planenden Bauwerks, dagegen ist weiterhin Domäne des Ingenieurs bzw. Architekten.
Obwohl technisch schlecht unterstützt, sind gerade die frühen Phasen des Entwurfs prägend für die Qualität und den Preis des Bauwerks sowie für die späteren Kosten für Betrieb, Modernisierung bis hin zur Entsorgung. Die Entscheidungen der frühen Planung determinieren zu ca. 90% die Gesamt*bau*kosten und dies zu einem Zeitpunkt, wo erst ca. 2% der Herstellungskosten durch die entsprechenden Planungskosten entstanden sind, (Abb. 1-2). Für die Gesamtkosten aus Herstellung, Nutzung und Entsorung über alle Bauwerkslebensphasen dürfte dieser Anteil noch erheblich höher liegen. Dies zeigt klar, wo die ökonomischen Potenzen des Computereinsatzes im Planungsprozeß liegen. So ist zwar der Planungsanteil an den Gesamt-(herstellungs)kosten auf bis zu 10% gestiegen [DOM95], eine Rationalisierung der Planungstätigkeit selbst bringt aber einen vergleichsweise geringen ökonomischen Gewinn, gemessen an einer qualitativen Verbesserung der Bauwerke selbst.

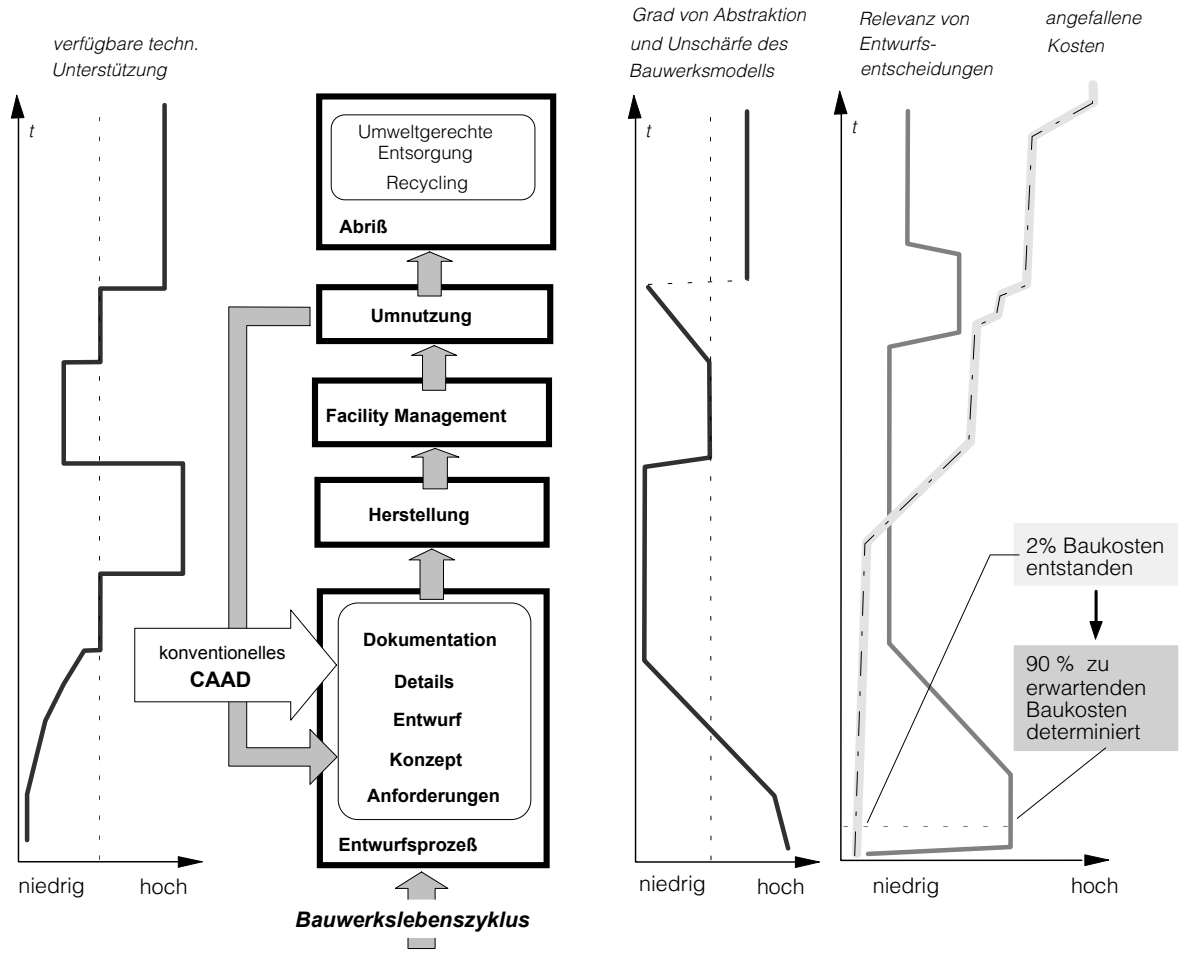


Abb. 1-2 Entscheidungsrelevanz und technische Unterstützung im Bauwerkslebenszyklus[9]

[9] in Anlehnung an [RIC88]

Die Gründe für die Diskrepanz liegen in der gänzlich anderen Aufgabenstellung in konzipierenden Entwurfsphasen (nicht algoritmierbare, synthetisierende Tätigkeiten) sowie dem Fehlen wissenschaftlicher Grundlagen (z.B. in der Künstlichen Intelligenz). Hinzu kommt eine tiefe Skepsis beim Nutzer, was den Einsatz von computerbasierten Techniken für innovative Designprozesse anbelangt sowie die starke Betonung menschlicher Intuition und Kreativität, deren psycho-soziale Hintergründe hier nicht erörtert werden sollen.

Auch für die Konzeption und Nutzung durchgängiger CAD-Systeme besteht großes Interesse an frühen Entwurfsphasen. In der Phase des konzeptuellen Entwurfs erfolgt die Konstituierung des Modellkerns durch das initiale Erzeugen, Benennen, Klassifizieren und Aggregieren von Entwurfsobjekten. Diese Struktur entwickelt sich und lebt fort über alle Lebensphasen des Bauwerkes. So lange das Bauwerk selbst existiert, besteht ein Bedarf zur Erhaltung seines informationstechnischen Abbilds. Es kann für die Organisation der Gebäudenutzung bis hin zur Unterstützung von Redesign- oder Recyclingprozesse genutzt werden (siehe auch [FIS94]). In [HOV94] wurde hierzu der Begriff des *'permanent designs'* geprägt. Die Modelldaten eines Bauwerks verkörpern daher auch einen beträchtlichen materiellen Wert. Man geht heute davon aus, daß der Datensatz zur Repräsentation komplexer Bürobauten, welcher als Grundlage eines computergestützten Facility Management dienen kann, ca. 2 % der Herstellungskosten wert ist[10]. Daten, die nur so lange interpretierbar sind, wie die sie erzeugenden Programme leben, sind dem langen Bauwerkslebenszyklus grob unangemessen. Während man bei Hard- und Software heute Umlaufzeiten von 2 bis 5 Jahren zu verzeichnen hat, liegen die geplanten Standzeiten für Hochbauten bei ca. 80 Jahren! Innerhalb dieser Zeit erfolgt häufig Umnutzung der Bauhülle verbunden mit einem Redesignprozeß[11].

1.3 Besonderheiten von Bauwerken als Gegenstand von Planungsprozessen

Der Planungsprozeß von Bauwerken unterliegt Besonderheiten, die aus dem spezifischen Charakter dieses Typus von technischen Produkten resultiert. Sie sind z.B. vergleichsweise teuer. Das benötigte Investitionsvolumen läßt sich nur mit technischen Großprojekten, etwa aus dem Schiffsbau, vergleichen. Bauwerke sind im allgemeinen Unikate und sei es nur durch die jeweils besonderen Standortbedingungen und die jeweils nötige Einzelbeantragung. Dies macht für jedes Bauwerk letztlich einen eigenständigen Planungsprozeß nötig. Selbst wenn Planungsunterlagen existieren, etwa für Typenbauten, so müssen diese speziell angepaßt werden. Dies ist ein Grund, warum eine derart präzise Planung, wie sie bei Autos etwa eine automatisierte Serienproduktion ermöglicht, für Bauwerke nur schwer vorstellbar ist. Die Herstellung erfolgt stark arbeitsteilig. Die beteiligten Gewerke besitzen einen relativ hohen Grad an Autonomie, wobei sie mit Planungsunterlagen arbeiten, die einen vergleichsweise geringen Detaillierungsgrad besitzen.

Auch der Planungsprozeß selbst erfolgt arbeitsteilig. Eine besondere Rolle kommt hierbei dem Architekten zu. Er leistet die ersten Phasen des Planungsprozesses, deren besondere Bedeutung für die Qualität des Bauwerks bereits hervorgehoben wurden. Seine Stellung wird durch das gültige Genehmigungsrecht weiter betont. Darüberhinaus organisiert und

[10] Der Prozentsatz stützt sich auf Angaben der Hochtief AG zum Commerzbank Tower Frankfurt/M.

[11] in [BER94] wird ausgeführt: Redesignzyklen für Wohnbauten 25 Jahre, für Banken und sonstige Bürobauten von 10 Jahren, Tankestellen/ Raststätten 5 Jahre, Industrieproduktionsstätten 2 Jahre, bis hin zu moderenen Produktionsanlagen der Elektronik von ¾ Jahren. Im letzten Fall heißt das, daß der Plansatz bei 80 Jahre Standzeit ca. 100 mal genutzt und fortentwickelt werden müßte!

überwacht er oftmals die Bauwerksfertigung im Rahmen der Autorenkontrolle. Durch die umfassende Präsenz des Architekten im Planungs- und Fertigungsprozeß ist der Architekt ein Generalist, dessen ideale Kompetenz Abbildung 1-3 umreißt.

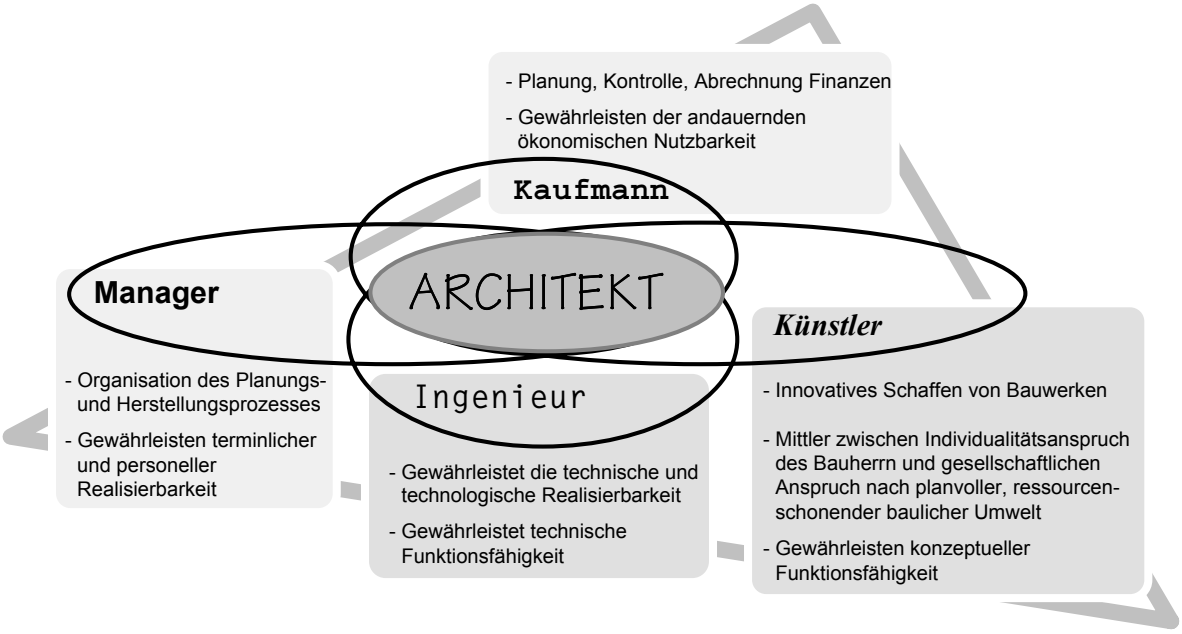


Abb. 1-3 Ideale Kompetenz des Architekten

In der heutigen Baupraxis kann ein Einzelner dem Leistungsanspruch, alle diese Arbeitsfelder umfassend bearbeiten zu können, nicht gerecht werden und so gibt der Architekt Verantwortung an Spezialisten ab. Die Rolle des Architekten ist für andere Industriezweige unbekannt und bildet sich im Moment mit der Funktion des 'Projektentwicklers' für technische Großprojekte heraus.

Ein weiteres Problem der Bauwerksplanung ist, daß die a priori-Bewertung von Bauwerken prognostischen Charakter mit einem weiten Zeithorizont trägt[12]. Sie gelingt nur qualitativ und ist dann nur subjektiv und unscharf ([HOV94], [GYA94]). Eine ökonomische Gesamtbilanz über den gesamten Bauwerkslebenszyklus wird nicht erstellt, so daß Rückkopplungen für neue Projekte nicht möglich sind. Aktuell wird dies exemplarisch für ausgewählte Bauwerke im Rahmen einer ökologischen Gesamtbilanz versucht, die Ergebnisse sind aber eher bescheiden ([BAR95], [MIH95]).

Wegen der genannten Probleme gibt es eine Reihe spezifischer Forderungen an CAD-Systeme im Architekturbereich und an die mit ihnen erzeugten Modelle. Im folgenden werden die Besonderheiten und die resultierenden Anforderungen aus der Sicht des frühen architektonischen Entwurfs aufgeführt:

(A) Bauwerke binden große Mengen an Ressourcen der Ökosphäre
- genaue Plan- und Bilanzierbarkeit erwünscht
- Modelleinordnung in regionale Planungsmodelle notwendig
- Verfahren des (prognostischen) Ökoaudits gewinnen an Bedeutung

[12] NIELS BOHR bemerkte 1949: „Prognosen sind schwierig - besonders für die Zukunft"

(B) Bauwerke sind Unikate
- hoher Anteil kreativer, nichtalgorithmierbarer Entwurfsaktivitäten
- wechselnde Planungsmethodik und Planungsziele
- Modelldatensatz ist ebenfalls Unikat
- Domänenmodelle besitzen hohen Abstraktionsgrad

(C) Bauwerke können nur qualitativ bewertet werden, die Bewertung ist subjektiv, sie sind oftmals Träger einer künstlerischen Aussage
- Bewertungen von Entwurfsentscheidungen tragen prognostischen Charakter
- Bewertungen unterliegen subjektiven Kriterien
- Algorithmen zur Berechnung komplexer Bewertungen existieren nicht
- Gütekriterien sind nicht immer skalar oder ordenbar
- Bewertungen müssen aus unvollständigen, unscharfen, abstrahierenden Modellen abgeleitet werden
- Eine 'Verrechnung' in ein umfassendes Gütekriterium, mit dessen Hilfe eine Halbordnung von Varianten möglich wäre, existiert nicht; multikriterielle Probleme sind nicht entscheidbar (es ist nur die Menge paretooptimaler Lösungen bildbar [PES80])
- automatische Entwurfsgeneratoren setzen Variantenbewertung; sie sind nur sehr eingeschränkt realisierbar

(D) Bauwerke sind (oftmals) universelle Strukturen
- Bauzweck liegt nicht im Bauwerk selbst begründet
- Teil des Planungsprozesses ist das Erfassen verschiedenster Nutzungstechnologien
- Bauwerksmodelle sollten in (wechselnde) Benutzungsmodelle integrierbar sein

(E) Bauwerke leben lange
- erhöhte Bedeutung der Planungsphase durch Kostendetermination über die lange Lebensphase
- rechnerinterne Modelle müssen tolerant gegen Änderungen von Hard-, Software und Speichermedien sein
- Modelle sollten auch in (weiter) Zukunft interpretierbar sein und bedürfen deshalb einer kognitiven begründeten Modellstrukturierung
- Prognoseverfahren/ -modelle für ökonomisches, ökologisches und bautechnisches Verhalten gewinnen an Bedeutung

(F) Bauwerke sind teuer
- Immobilien müssen gesondert bewirtschaftet werden, die Erweiterung von Modellen um ökonomische Informationen muß zur Zeit der Nutzung möglich sein; die Modelle benötigen die Zeit als eine Objektkoordinate (z.B. Wartungsfristen)
- Datensatz des Modells stellt ebenfalls einen beträchtlichen Wert dar

(G) Bauwerke werden häufig umgenutzt
- Modelle sollten auch in (weiter) Zukunft interpretierbar sein und bedürfen deshalb einer kognitiven begründeten Modellstrukturierung
- Modelle müssen tolerant gegen Änderungen von Hardware, Software und Speichermedien sein
- Es bestehen Probleme der Konsistenzsicherung zwischen Modell und Realität
- Bauwerksmodelle sind ihrem Charakter nach 'lebende' Modelle

(H) Bauwerke werden arbeitsteilig geplant[13]
- Der Kommunikationsbedarf ist hoch, da im allg. eine räumliche und zeitliche Trennung der Bearbeitergruppen vorliegt
- Das Erstellen spezieller Repräsentationen ist nach wie vor Standard und bedingt großen personellen und zeitlichen Aufwand
- Modellwelten sind nicht disjunkt (multipler Modellzweck)
- Konsistenz der Partialmodelle ist nur schwer zu sichern

(I) Bauwerke besitzen Planungsunterlagen mit vergleichsweise geringer Detaillierung
- spezielle Domänenmodelle der Gewerke enthalten vergleichsweise viele Implikationen
- Unschärfe und Abstraktion sind immanente Modelleigenschaften

(J) Bauwerke werden arbeitsteilig hergestellt
- Für die Koordination der Herstellung muß eine Kontrollinstanz existieren, die spezielle Planungsmodelle benötigt (Modellierung von zeitbezogenen Verhalten)
- Es ist fragwürdig, in wieweit Planungsmodelle a posteriori als Beschreibung der gebauten Realität Verwendung finden können (Konsistenz Modell - Realität)

(K) Bauwerke werden am Nutzungsort hergestellt (wandernde Produktionsstätte)
- Die Bauwerksfertigung bedingt eine hohen logistischen (Planungs-) Aufwand
- Lokale Besonderheiten haben großen Einfluß auf Planungsprozeß und Modelle

Die Konsquenzen aus den getroffenen Feststellungen zu Bauwerken als Planungsgegenstand sowie der besonderen Bedeutung der frühen Phasen in diesem Prozeß werden im folgenden noch einmal zusammengefaßt.

(1) Eine Unterstützung von frühen, durch den Architekt zu leistende Entwurfsphasen, ist sinnvoll und notwendig. Diese Phasen prägen in hohem Maße Qualität, Ökonomie und Ökologie des Produkts Bauwerk, dessen Herstellung und Betrieb große Ressourcen bindet.

(2) Bauwerke werden in einem hochgrad arbeitsteiligen Prozeß geplant und hergestellt. Dies bedingt, daß eine möglichst verlustfreie Übertragung von Informationen in besonderem Maße Ziel einer CAD-Systemstützung sein muß.

(3) Die Bautechnologie ist universell. Sie schafft mit dem Gebäude ein Produkt, das verschiedenen, oft wandelbaren Nutzungszwecken an einem jeweils spezifischen Nutzungsort genügen muß. Dies bedingt, daß jeder Planungsprozeß einzigartig ist. Er ist durch eine Menge von CAD-Tools zu stützen, die problemspezifisch zum CAD-System konfiguriert werden.

[13] arbeitsteilig bezieht sich auf die Zusammenarbeit verschiedener Ingenieurgewerke, die in selbständigen wirtschaftlichen und strukturellen Einheiten organisiert sind, dies gilt analog für Punkt *(J)*

2. CAAD – WERKZEUGE IM BAUPLANUNGSPROZEß

Als zu Beginn der 60er Jahre die hard- und softwareseitigen Voraussetzungen gegeben waren, wurde begonnen computergestützte Planungshilfen für Architekten zu entwickeln. Gemäß dem damaligen Zeitgeist, der durch die Vorhersagen von NEWELL und SIMON geprägt wurde, waren die Ziele hochgesteckt. Es wurde nach Verfahren gesucht, die die Erzeugung (!), Bewertung und Darstellung von Entwurfslösungen erlauben sollten. Ein markantes Beispiel dieser Versuche ist etwa in NEGROPONTE "The architecture machine" (1970, [NEG70]) zu finden[1]. LIEBIG [LIE93] ordnet diese Versuche in eine 'Erste Generation' von CAD-Systemen ein und benennt zwei weitere Generationen. Der Begriff 'Wellen der CAD-Systementwicklung' dürfte allerdings angebrachter sein, wenn man die jeweilige Aufeinanderfolge von Euphorie und Ernüchterung betrachtet.

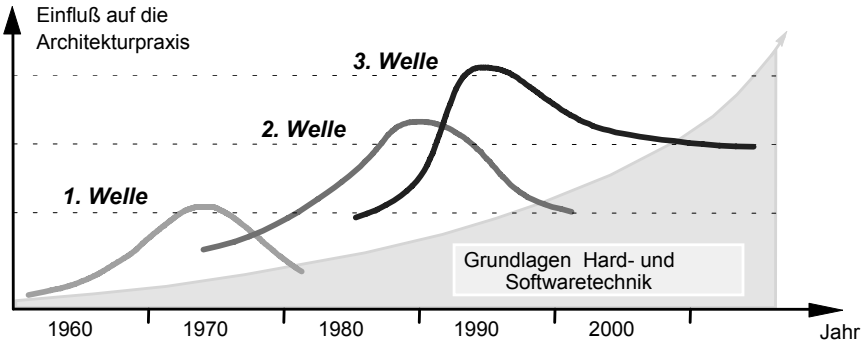


Abb. 2-1 Wellen der CAD-Systementwicklung

Die zweite Welle brachte nach der eingetretenen Ernüchterung anwendungsneutrale Standardsoftware wie Datenbanken, Statiksoftware und, ausgehend von neuer Peripherie, interaktive Zeichnungseditoren und Geometriemodellierer bis hin zur fotorealistischen Darstellung geometrischer Modelle. Diese Anwendungspakete in Verbindung mit dem Aufkommen leistungsfähiger PC's brachten Anfang der 80er Jahre auf breiter Front den Einzug von Computertechnik in Architekturbüros[2]. Waren es anfangs lediglich die Verwaltung und Organisation numerischer und textueller Informationen, so werden gegenwärtig auch das Erstellen, Verwalten und Präsentieren grafischer Informationen weitgehend unterstützt. Diese Anwendungen sind vom Anspruch eines Denkzeuges jedoch weit entfernt. Infolge der historischen Entwicklung handelt es sich bei CAD heute nicht um **C**omputer **A**ided ***Design*** sondern eher um **C**omputer **A**ided ***Drafting***, wie auch die Dominanz von zeichnungsbasierten Schnittstellen wie DXF[3] als (Daten-) Austauschformat belegt.

Der Computer kombiniert und erweitert als universelles Arbeitsmittel die Fähigkeiten

- einer elektronischen Schreibmaschine (Textverarbeitung, DeskTop Publishing)
- eines Ordner- und Ablagesystemes (Projektverwaltung, Datenbanken)
- eines erweiterten Taschenrechners (Tabellenkalkulation, Berechnungsprogramme)
- eines elektronischen Reißbrettes (Zeichnungs- und Präsentationsprogramme).

[1] Die einsetzende Ernüchterung bzw. Umorientierung bei den Anforderungen an CAD-Systeme verdeutlicht "Total Digital" [NEG95] desselben Autors

[2] Eine Recherche von DONATH/MAYE von 1991 [DON91] gibt an, daß in Thüringen 50% aller Architekturbüros über Computertechnik verfügen. Der Schwerpunkt der Nutzung lag allerdings mit ca. 63%. bei AVA und Textverarbeitung, während damals nur ca. 16% CAD einsetzten. Bedingt durch den Druck von Auslobern und Auftraggebern zu computererstellten Entwurfsunterlagen und Präsentationen dürfte heute dieser Prozentsatz wesentlich höher liegen.

[3] 'Drawing Exchange Format' - proprietäres, ASCII-basiertes Zeichnungsaustauschformat von AutoDesk

Rein zeichnungsorientierte CAD-Tools werden gegenwärtig durch Systeme abgelöst, die ein rechnerinternes (3D-) Modell des zu planenden Gebäudes erstellen. Die dreidimensionale Beschreibungsmöglichkeit bietet zwar eine wesentlich genauere Charakteristik des Bauobjekts, doch auch hier werden im allgemeinen lediglich geometrische Daten verarbeitet. CAD-Systeme sind heute aber auch in der Lage, mit den Geometriedaten weitere, nichtgeometrische Informationen zu verbinden[4]. Damit könnten alle bauwerksrelevanten Daten in CAD-Systemen erfaßt werden, soweit sich geometrische Elemente finden lassen, zu denen sie zuordenbar sind.

Bei der Bauwerksbeschreibung durch heute erhältliche CAD-Systeme bleiben folgende Mängel bestehen:

- Die Beschreibung erfolgt vorzugsweise über die Geometrie, d.h. Informationen können nur mit geometrischen, grafisch repräsentierbaren Elementen verbunden werden. Das Primat der Modellabbildung sollte das Objekt an sich haben, seine Geometrie bzw. spezifischen Darstellungsarten sind lediglich Aspekte des Objektes.

- Heutige CAD-Programme sind neuen Anforderungen ihrer Nutzungsdomäne nur schwer anpaßbar. Ändern sich Anforderungen an die Software, wie etwa Vorschriften bzw. Richtlinien für das Anwendungsgebiet, müssen Programmierer eingreifen und die Software entsprechend verändern. Dasselbe gilt für neue, nicht vorgedachte Entwurfselemente. Besser wäre es, wenn solche Änderungen, soweit als möglich, vom Anwender dem System mitgeteilt werden könnten, ohne dabei programmierend eingreifen zu müssen.

- Die Nutzung heutiger CAD-Systeme erfordert immer genaue Beschreibungen. Dieser Zwang zur Genauigkeit erschwert konzipierende Handlungsweisen, wie sie vor allem beim Entwerfen in frühen Phasen auftreten. Unschärfe und Abstraktion als inhärente Eigenschaft früher Entwurfsmodelle werden nicht toleriert. Entsprechend wird häufig eine ‘Pedanterie des CAD Handlings’ beklagt ([GI91]).

- Bauwerke oder deren Teile lassen sich nicht auf verschiedenen Abstraktionsstufen betrachten. So ist es beispielsweise nicht möglich, am Anfang des Entwurfsprozesses Angaben zum Bauwerk als Ganzes zu machen, ohne dessen Teile zu kennen. Existieren keine solchen höheren Aggregate, gibt es im System auch keine Objekte, mit denen sich die entsprechenden Informationen assoziieren lassen.

Die CAD-Systeme arbeiten heute auf einer von der Programmierung vorbestimmten Sicht auf Bauwerke. Aus wirtschaftlichen Gründen müssen sie im allgemeinen von vielen Anwendern genutzt werden. Bei ihrer Nutzung kann also nur eine gemeinsame Grundmenge von konsensfähigen Begriffen, Konzepten und Arbeitsweisen verwandt werden. Sie sind zweckgebunden nur für Entwurfsarbeiten nutzbar, die dieser Sicht und dem zur Programmierzeit aktuellen Kenntnis- und Vorschriftenstand entsprechen. Ein CAD-System gibt damit praktisch eine methodische Arbeitsweise vor. Der Anwender entscheidet sich durch die Benutzung des Systems für die darin enthaltene Methodik.
Besonders in frühen Entwurfsphasen zeigt sich, daß dies nur dann akzeptiert wird, wenn dem Entwerfenden genügend Spielraum zur Verwirklichung seiner Entwurfsintensionen und zu kreativem Arbeiten bleibt. Die entsprechende Entwurfsmethodik darf keine feste Schrittfolge vorschreiben, sondern sie sollte lediglich die Randbedingungen für eine systematische(re) Arbeitsweise bereitstellen. Dies impliziert die schrittweise Konkretisierbarkeit von Entwurfs-

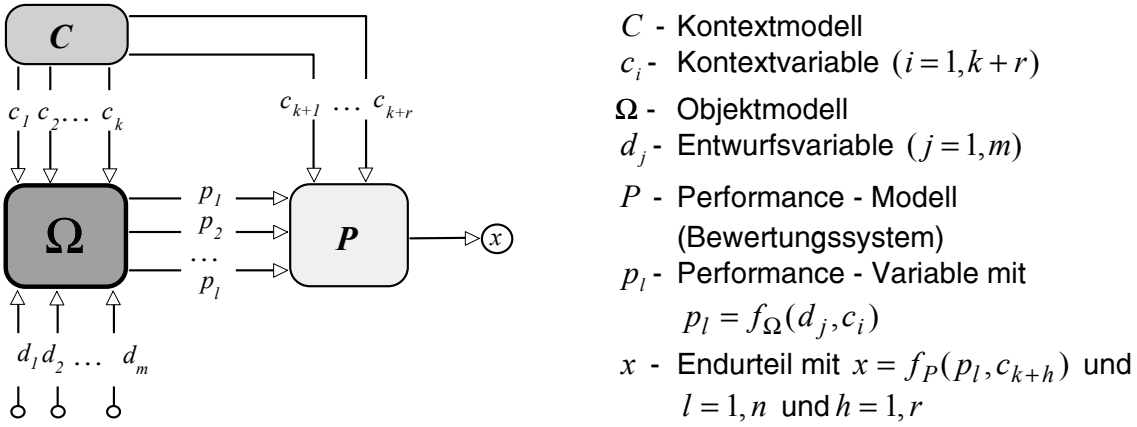
[4] Eine Variante ist der aus dem Maschinenbau übernommen Feature-Ansatz [STA92], der stark vereinfachend mit der Formel *Feature = Geometrie + Semantik* beschrieben wird.

objekten sowie deren leichte Änderbarkeit bei breiter Nutzung der analytischen Möglichkeiten des Computers. Voraussetzung ist, daß man das Gebäude systematisch so gliedern kann, wie es seinem Aufbau aus Einzelteilen entspricht. Obwohl der Nutzer mental immer vom Groben zum Feinen, vom Ganzen hin zu seinen Teilen entwirft, erfolgt die Bearbeitung in heutigen CAD-Systemen umgekehrt. Man muß bei der CAD-Nutzung mit irgendeinem mehr oder weniger genau zu beschreibenden Teil beginnen. Das Gebäudekonzept muß dabei bereits im Kopf des Entwerfenden vorliegen, d.h. er hat für sich (ohne Computerstützung) das Entwurfsproblem zumindest partiell gelöst.

2.1 Klassen von Entwurfsproblemen

Für die Beschreibung von Entwurfs*problemen* wurden verschiedene Formalisierungen vorgeschlagen, die letztlich alle ihre Wurzeln im 'Operation Research' haben. Entsprechende Ansätze sind in [GER90], [DÖR93], [TAN93a], [GER95A], [RIT92] zu finden, wobei auf die letzte Quelle hier exemplarisch Bezug genommen wird.

Zentraler Bestandteil der Problembeschreibung bei RITTEL ist das **Objektmodell** Ω, hier in Form eines Vektors $\bar{d}$ von Modellparametern. Bestimmt wird das Problem durch einen Entwurfskontext C (auch Randbedingungen). Aus der Konfiguration des Objektmodells, d.h. einem konkreten $\bar{d}$ folgt ein Vektor von Performance - Variablen $f_\Omega(\bar{d},\bar{c}) \rightarrow \bar{p}$. Da hier ein Vektor von Bewertungen (Performance) vorliegt, handelt es sich um eine mehrkriterielle Entscheidung. Dieser Entscheidung durch Auswahl einer (Objektmodell-) Variante aus der Varietät erzeugbarer Varianten liegt ein Performance - Modell P zugrunde. Die Anwendung dieses Modells auf $\bar{p}$ liefert eine Bewertung x auf einer Ordinalskala $f_P(\bar{p},\bar{c}) \rightarrow x$.



C - Kontextmodell
c_i - Kontextvariable $(i = 1, k+r)$
Ω - Objektmodell
d_j - Entwurfsvariable $(j = 1, m)$
P - Performance - Modell (Bewertungssystem)
p_l - Performance - Variable mit $p_l = f_\Omega(d_j, c_i)$
x - Endurteil mit $x = f_P(p_l, c_{k+h})$ und $l = 1, n$ und $h = 1, r$

Abb. 2-2 Kontext - Objekt - Performance Modell (KOP-Modell) , angelehnt an [RIT92]

Der Wert einer solchen sehr abstrakten Beschreibung eines Entwurfsproblems liegt nicht in ihrer mittelbaren oder gar unmittelbaren Nutzbarkeit zur Darstellung konkreter Probleme, sondern in der Charakterisierbarkeit von Klassen von Entwurfsaufgaben im allgemeinen. Diese Klassifizierung gibt Hinweise, ob und in welcher Weise Computertechnologie zur Problemlösung eingesetzt werden kann. Lösungsverfahren und die zu erwartende Problemkomplexität lassen sich ggf. ableiten[5].

[5] Anmerkung: Die Komplexitätstheorie liefert hierzu Rüstzeug, eine geschlossene Analyse realer Probleme dürfte allerdings unmöglich sein. In [DON88] wird der Entwurfsprozeß systemtheoretisch als '*Offenes System*' bezeichnet, das sich unter Einwirkung irgendwelcher Art verändert oder als wandelbar angenommen wird.

In der Literatur werden im wesentlichen 3 Klassen von Entwurfsproblemen unterschieden:

• *Well-Defined Problems*	• *Ill-Defined Problems*	• *Wicked-Defined Problems*	bei [ROW87]
• *Class3-Design*	• *Class2-Design*	• *Class1-Design*	bei [CHA89]
• *Routine-Design*	• *Innovative-Design*	• *Creative-Design*	bei [KAR90]

Da sich die Klassifikation nach ROWE [ROW87] direkt an architektonischen Entwurfsproblemen orientiert, sei sie hier ausführlicher vorgestellt:

- WELL-DEFINED PROBLEMS (Routine-Design)

Dies sind Probleme, für die das Ziel ihrer Lösung klar definiert ist, d.h. es muß entscheidbar sein, ob eine Lösung oder ob gar eine optimale Lösung vorliegt. Lösungselemente und Lösungsverfahren sind vollständig bekannt. Es werden jedoch keine Aussagen über die zur Lösung benötigten Ressourcen getroffen. RITTEL charakterisiert dies wie folgt: Hierbei handelt es sich um ein Problem „... das vollständig beschrieben ist und das von einem entsprechend geschulten Menschen ohne weitere Informationen gelöst werden kann“. NEWELL, SIMON und SHAW formulieren: „gegeben ist eine Menge von Elementen A (Menge möglicher Lösungselemente), gesucht ist ein Subset $B \,|B \subseteq A$ (B - Lösungselemente), für deren Elemente spezifische Eigenschaften gelten (Randbedingungen).“.
Beispiele dieses Problemtyps wären lösbare lineare Gleichungssysteme, Kreuzworträtsel oder Brettspiele. Hauptproblem dieser Aufgabenklasse ist häufig die zu verzeichnende kombinatorische ‘Explosion’. In Architektur und Stadtplanung handelt es sich um Probleme wie das Aggregieren, Positionieren und Dimensionieren von Bauwerken bzw. Bauwerksteilen und deren Entwurf, soweit er vollständig durch ein Regel- oder Vorschriftenwerk determiniert ist.

- ILL-DEFINED PROBLEMS (Innovative-Design)

Sowohl Lösungsverfahren und Lösungselemente als auch die Entwurfsziele sind zu Beginn des Problemlösungsprozesses nur unvollständig bekannt (NEWELL, SHAW, SIMON). Die meisten Entwurfsprobleme in Architektur und Stadtplanung sind von diesem Typ. Wenn ein Bauherr einen Entwurfsauftrag vergibt, hat er im allgemeinen nur vage und abstrakte Wünsche (Entwurfsziele). Ein beträchtlicher Teil des Problemlösungsprozesses befaßt sich mit der Formulierung von Entwurfszielen und -randbedingungen und damit mit der Problemdefinition und Redefinition. Die in diesem Prozeß gefundenen Lösungen sind Lösung des aktuellen Problems, das vom initialen Problem verschieden sein kann. Ständige Lösungsrevision ist somit ein grundsätzliches Vorgehen für diese Problemklasse. Durch die inhärente Verflochtenheit von Problemstellung und Lösungszustand handelt es sich bei Entwurfsprozessen systemtheoretisch i.allg. um ‘offene Systeme’.
Beispiele hierfür sind der Entwurf von Bauwerken, die den Charakter echter Unikate tragen oder ein umfassendes Redesign von Bauwerken. Die Lösung ist nicht ausschließlich durch Vorschriften determiniert, sondern durch allgemeine Regeln der Baukunst. Es liegen keine algorithmischen Verfahren zur Lösungsfindung vor.

- WICKED PROBLEMS (Creative-Design)

Hier handelt es sich um Probleme, die so unzureichend definiert sind, daß der eigentliche Problemlösungsprozeß in den Hintergrund tritt (CHURCHMAN 1967, RITTEL 1972).

Für diese Probleme gilt:

- Es liegt keine endgültige Aufgabenstellung vor und sie wird wahrscheinlich auch nicht erstellbar sein. Es lassen sich immer neue Aspekte und Fragestellungen finden, die zu einer geänderten Aufgabenstellung führen können.
- Es gibt keine Regeln, die den Entwurfsprozeß terminieren lassen, d.h. es ist nicht entscheidbar, ob ein 'Endergebnis' vorliegt.
- Auch eine kleine Veränderung der Aufgabenstellung führt zu anderen Entwurfsergebnissen, d.h. der Prozeß der Problemdefinition bestimmt letztlich das Ergebnis.
- Für eine Lösung läßt sich nicht sicher feststellen, ob es eine gültige oder gar gute Lösung ist. Es liegen immer Alternativlösungen vor. Zeitweilige Ergebnisse unterliegen einer Bewertung und anschließender Problemredefinition. Der Problemlösungsprozeß ist ein iterativer Zyklus, der meist nur aus Gründen beschränkter Lösungszeit terminiert.

Im Rahmen des Bauwesens trifft diese Problemstellung auf echte Unikate zu, die einem erhöhten künstlerischen Anspruch genügen sollen oder für die eine neuartige Konstruktion oder Nutzung vorliegt.

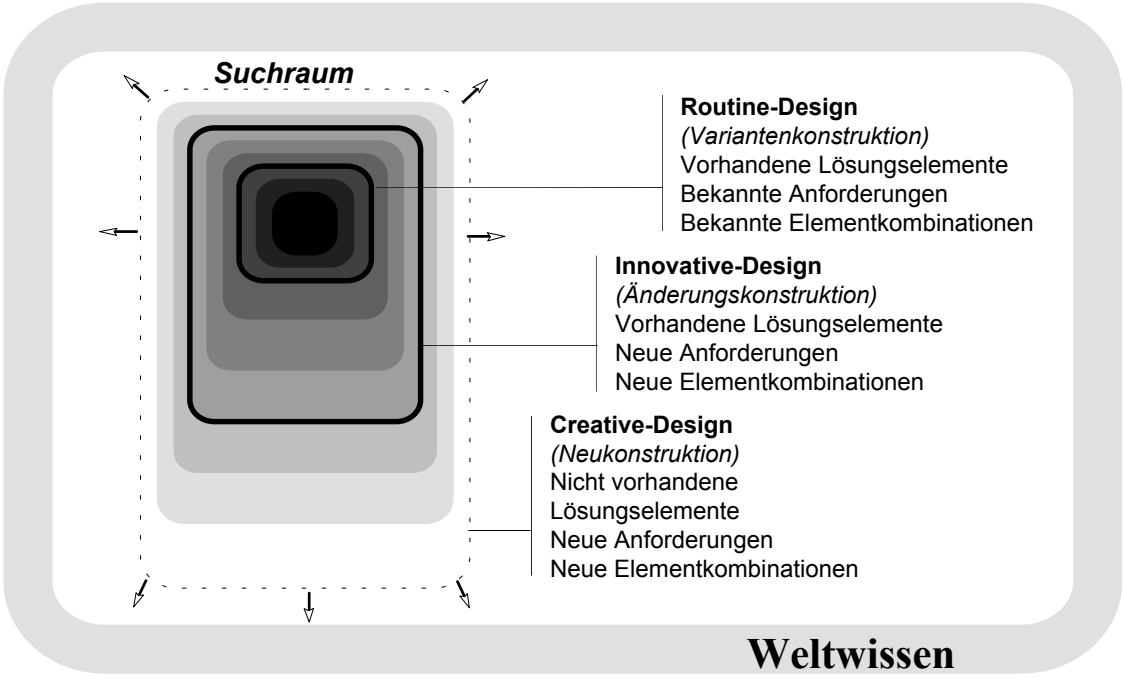


Abb. 2-3 Einordnung Designprobleme

In [GER95b] wird GARDNER (1993) zitiert: „... die Menschen, die wir als am meisten kreativ erachten ... ändern tatsächlich das *Wesen* ihrer Domäne. Die Konsequenz ist, daß die nächste Generation von Menschen jetzt eine Domäne zu untersuchen hat, die anders zusammengesetzt ist."

Dies ist gewiß die höchste Form von Kreativität, sie wird in [GER89] als 'epochales Design' bezeichnet. In diesem Fall ist der Suchraum für neue Lösungen nur durch grundlegende Naturgesetze beschränkt. Durch den Lösungsprozeß wird das Weltwissen selbst erweitert.

Der Grad der Unterstützbarkeit von Entwurfsproblemen durch Computer hängt von der Aufgabenklasse ab. Eine Unterstützung im Sinne eines 'Entwurfsgenerators' mit dem Ziel des automatischen Herleitens von Entwurfslösungen aus der Aufgabenstellung gelingt gegenwärtig nur für spezielle Probleme des 'routine designs', wie etwa der Layoutplanung

oder einfacher Grobgeometrieerzeugung. CHANDRASEKARAN ([CHA89]) bezeichnet diesen Problemtyp als „... an excellent place to start in an attempt to fully understand the complete spectrum of design activity“, womit dann auch der nach wie vor gültige ‘state-of-the-art’ der Computerstützung umrissen ist.

Für die permanente, systematische, computergestützte Erfassung einer Entwurfsaufgabe liegen wenig Erfahrungen vor. Eine erste Möglichkeit ist die unstrukturierte Sammlung von Informationen zur Bedarfspräzisierung und Kontextanalyse, meist in multimedialer Form wie etwa in [TSO95] und [MAV94]. Es sollte versucht werden, die erfaßten multimedialen Dokumente in das eigentliche Objektmodell einzubinden und so eine Kopplung zwischen Kontextmodell und Objektmodell zu realisieren.
Der zweite Weg liegt in einer strukturierten, formalisierten Darstellung des Kontextes, wie er in [MAT91] , [MED95], [CAR94] vorgestellt wird. Leider wird aus den angeführten Veröffentlichungen nicht deutlich, ob die Strukturierungsmöglichkeiten fixiert sind, d.h. daß nur definierte, vorgedachte Modellelemente instanziiert werden können, was eine starke Einschränkung der Ausdrucksmöglichkeiten bedeuten würde. Die Bedarfs- und Kontextanalyse ist ja gerade dadurch geprägt, daß sie neue, nicht dem primär baulichen Kontext zugehörige Modellelemente integrieren muß. Um die dynamischen, unscharfen und subjektiven Modellwelten, wie sie für Kontext- und Bedarfsanalyse vorliegen, zu unterstützen, müssen neue Typen von Modellelementen innerhalb des Modellierungsparadigmas erzeugbar sein, d.h. derartige Tools sollten nicht a priori auf eine Domäne oder die Intentionen des Programmierers fixiert sein.

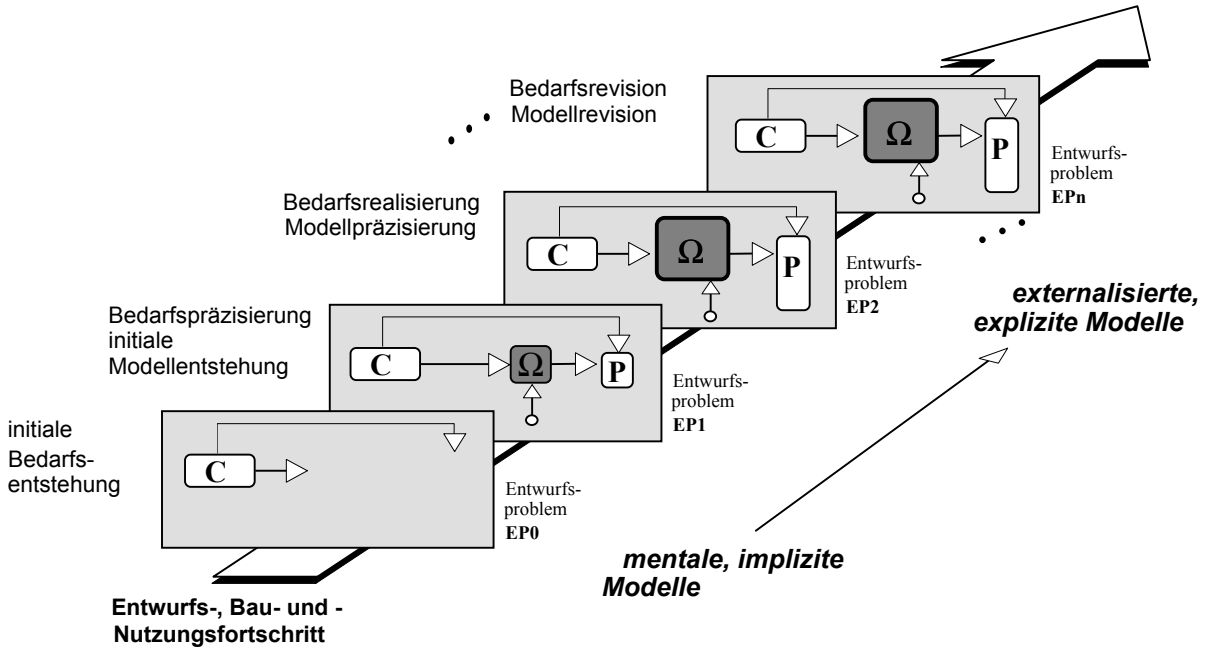


Abb. 2-4 Dynamik des Entwurfs*problems*

Es ist anzuerkennen, daß im Rahmen des ‘*offenen Systems’* Entwurf der Modellaufbau durch die „Verflochtenheit von Zielermittlung und Lösung“ ein iterativer Prozeß ist, der „von einer Lösungsstufe niederen Niveaus zu einer ... höheren Niveaus gelangt“ [DON88]. Jeder Problemzustand ist also an einen konkreten Zeitpunkt gebunden. Diese Dynamik bei der Entwicklung des Entwurfsproblems gilt es weitgehend zu unterstützen.

2.2 Klassen von CAD-Systemen

Bei der inhaltlichen Erstellung des konzeptuellen Entwurfs bieten heutige CAD-Systeme kaum Unterstützung ([GI91], [GYA94], [HÜB95]). Die Gründe hierfür sind hoher Abstraktionsgrad, Unschärfe, Unvollständigkeit, hohes Maß an Subjektivität bei Vorgehen, Annahmen und Modellinterpretation. Der Entwurfsprozeß und die in ihm erbrachten kreativen Entwurfsleistungen sind gegenwärtig unverstanden. In den aktuellen, letztlich auf Symbolen basierenden Programmierparadigmen, entziehen sie sich der Algorithmierbarkeit. Die Menge denkbarer Entwurfsobjekte ist unbegrenzt oder besser unbegrenzt differenzierbar, gleiches gilt für die Menge der Relationen über diese Objekte. Es muß also für realistische architektonische Entwurfsaufgaben von einer unbegrenzten Komplexität ausgegangen werden. Für die Lösung derartiger Planungsaufgaben kommt dem Entwerfenden (Mensch oder Maschine) nur eine begrenzte Kompetenz zu ([AYE91]). Für generative CAD-Systeme, also solche, die aktiv in einem gegebenen Problemraum eine Lösung generieren, muß dieser Problemraum begrenzte Komplexität besitzen. Aus der Künstlichen Intelligenz ist dies unter der Bezeichnung '*closed world assumtion*' bekannt ([PUP91]). Alle anderen Systeme tragen den Charakter passiver bzw. assistierender Systeme. Sie bieten bei einer spezifischen Problemstellung angepaßte Beratungskompetenz sowie Kontrollkompetenz, die die Konsistenz des Entwurfs sichert, indem sie im Sinne eines Hintergrundprozesses Entwurfsentscheidungen des Architekten begleitet.
Basis der folgenden Klassifikation von CAD-Tools ist die Frage, ob sie toolspezifische Modelle besitzen und wenn ja, welche Art von Funktionalität sie über diese Modelle anbieten.

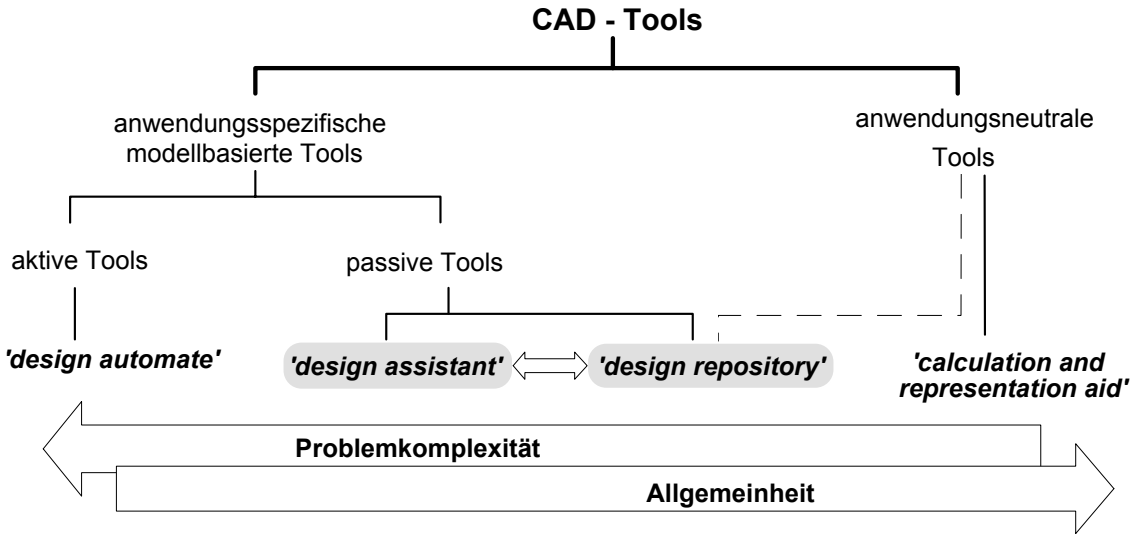


Abb. 2-5 Klassifikation CAD-Tools

(A) 'REPRESENTATION AND CALCULATION AID'

Hierbei handelt es sich um Entwurfs-*Hilfsmittel*. Die Programme verfügen über keinerlei spezifische Domänenmodelle. Sie ver- bzw. bearbeiten abstrakte Elemente wie Linien, Flächen, Körper, Matrizen, Zahlen oder Strings ohne anwendungsspezifische Semantik. Beispiele hierfür wären Programme zur Textverarbeitung, Tabellenkalkulation, zeichnerischen Konstruktion (AutoCAD), fotorealistischen Visualisierung (3D-Studio), allgemeine Statistik- oder Optimierungsprogramme. Diese Systeme sind die gegenwärtig dominierende Form der Computernutzung im Architekturbereich.

(B) 'DESIGN ASSISTANT'

Dies sind Tools, die über ein Produktmodell der behandelten Domäne verfügen. Sie unterstützen den Nutzer beim Aufbau konkreter Bauwerksmodelle als Instanzen solcher Produktmodelle. Die Systeme liefern Funktionen zur Plausibilitäts- und Konsistenzsicherung von Modellen sowie zu ihrer Bewertung auf der Basis eines meist impliziten 'Performance'-Modells (Modellevaluatoren). Der Umfang und die Flexibilität der Domänenmodelle variiert dabei jedoch sehr stark. Sie sind i.allg. auf den Entwurf eines speziellen Konstruktionstyps in einer spezifischen Entwurfsphase fixiert. Beispiele wären 'A4' - Entwurf hochinstallierter Gebäude in MIDI-Bauweise nach der Armilla-Methodik ([HAL85], [HOV91]), 'BOCAD' - Konstruktion von (Stahl-) Stabwerken, MCDA (Auswahl und Bewertung von Behälterkonstruktionen [GYA92]), 'Palladio' - sensitives Entwurfstool für den Hochbau mit wechselndem Detaillierungsgrad. Die genannten Tools sind teilweise im Einsatz, ihre Verbreitung ist aber noch relativ gering (Tendenz steigend).

(C) 'DESIGN AUTOMATES'

Diese Tools verfügen neben einem Produktmodell auch über Möglichkeiten zur Wissensdarstellung. Dieses Wissen kann zur Plausibilitäts- und Konsistenzsicherung eingesetzt werden sowie zur Vorgehensplanung und ansatzweise zum Repräsentieren von Entwurfskontexten. Sie sollten wechselnde Entwurfskontexte sowie unscharfe oder fehlende Umgebungsparameter verkraften. Grundlage zur Entscheidung über Entwurfsvarianten sind allgemeine Kriterien der Wirtschaftlichkeit oder formal ästhetische Gesetzmäßigkeiten. Ein algoritmisches Vorgehen zur Lösung solcher Aufgaben fehlt, sie bedienen sich daher oft einer schwachen Problemlösungsstrategie (Generiere-Teste-Strategie, [PUP91]).

Im Rahmen der getroffenen Unterscheidung von Entwurfsproblemen läßt sich die Unterstützung durch Tools der Kategorie C ('Design Automat') noch einmal in zwei Problemfelder mit unterschiedlicher Komplexität unterteilen:

(C1) MODELLGENERATOREN

Bei diesen Tools erfolgt ein generativer Aufbau eines Objektmodells. Die Komplexität kann beherrscht werden, indem die Menge von instanziierbaren Elementtypen stark reduziert wird. So gehen Systeme vor, die auf einfachen Geometrien arbeiten wie etwa im Abb. 2-6a und 2-6b. Der andere Weg zur Komplexitätsbeherrschung ist die Verwendung einer 'starken' Strategie zur Bildung komplexer Objektmodellstrukturen, d.h. deren weitgehend algorithmische Erzeugung. Diese Strategie könnte beispielsweise eine 'shape grammar' sein, wie sie im System 'CODE' (Generieren von Grundrißlayouts nach einer Formengrammatik, LIEBIG [LIE93]) Verwendung finden oder wie sie FLEMMING zur Generierung von 'Queen Anne'-Häusern anwendet (in [FLE94], Abb. 2-6a).
Weitere Beispiele solcher Programme sind 'HIRIS' (konzeptueller Entwurf von Hochhäusern ([MAH85]), 'TLE' (Vorentwurf von Stahlbauträgerlagen ([STE90]), 'LOOS' (generatives, wissensbasiertes System für die Synthese rechtwinkliger Grundrißlayouts [FLE90a]), 'GENESIS' (regelbasiertes System zur Generierung von 3D-Konfigurationen [HEIS93]). Alle aufgeführten Programme sind Forschungsprototypen. Dem Autor ist kein generatives Entwurfstool bekannt, das eine breite Nutzung in der Praxis erfahren hätte. Der Grund hierfür ist, daß die vorgestellten Programme nur für den formorientierten Entwurf in sehr eingeschränkten Domänen Verwendung finden können. Sie arbeiten nur, wenn der Entwurf als '*kanonisch*' (vergl. BROADBENT [BRO73]) bezeichnet werden kann. Dies bedeutet, daß regelhaftes Wissen verfügbar ist, mit dem Entwurfsvorschläge generiert und bewertet werden

können. So formulierte RITTEL bereits 1978: „Es ist nicht schwierig, eine 'Mies-Maschine' oder einen 'Palladio Simulator' zu programmieren“ – aber eben nur dann, wenn der Entwurf als kanonisch angesehen werden kann, *kreativ* im Sinne der Problemklassifikation ist dies sicherlich nicht.

(C2) MODELLVARIATOREN

Bei diesen Tools erfolgt kein eigentlicher Aufbau eines Objektmodells, sondern es werden 'lediglich' dessen Modellparameter variiert. Layoutprobleme gehören in diese Problemklasse, wie das Erzeugen von Kabinenlayouts des Airbus A340 [KOP93], das System 'WRIGHT' für Küchenlayouts [BAY90] oder Problemstellungen wie sie etwa in 'LOOS' ([FLE90a]) behandelt werden (Abb. 2-7).

Die Klassifikation eines CAAD-Tools in die drei genannten Gruppen muß im Einzelfall keine klare Zuordnung ergeben. So verfügen Systeme der Kategorie C oftmals über Merkmale (und Ansprüche) von 'Design Assistants' (siehe System 'SEED' [FLE96]). Allgemeine Tools der Kathegorie A arbeiten zunehmend über Modellen, die sich zwar nicht an speziellen Domänenmodellen orientieren, aber eine spezielle funktionale Sicht realisieren, wie sie in vielen Domänen benötigt wird. Bei der Kopplung derartiger Tools wird dann durchaus begrenzte Kompetenz erlangt, d.h. die Tools migrieren ebenfalls in Richtung 'Design Assistants'. Beispiele wären allgemeine Tools zur FEM-Berechnung in Verbindung mit Nachweismodulen oder die Massenermittlung in allgemeinen Geometriemodellierern kombiniert mit Programmen zur Ausschreibung.

In dieser Arbeit wird jedoch ein anderer Weg der Migration in Richtung hin zu Entwurfshilfsmitteln vorgeschlagen. Das heute übliche Vorgehen spezialisiert Modelle gemäß eines Modellzwecks, um komplexe analysierende, repräsentierende oder generative Algorithmen über den Modellen arbeiten zu lassen. Legt man jedoch keinen spezifischen Modellzweck zugrunde, genügt es, lediglich ein generisches Modellstrukturierungsparadigma vorzugeben. Innerhalb dieser Strukturierungsvorschrift kann der Nutzer dynamische Modellwelten schaffen, von denen er meint, daß sie seine Modellintentionen angemessen repräsentieren. Beim gegenwärtigen Stand der Softwaretechnologie auf der Basis der starren 'Modellieren⇨Compilieren⇨Nutzen'-Sequenz ist deren algorithmischer Gehalt notwendigerweise eingeschränkt und erstreckt sich nur auf statische Modellelemente, die als Basis dieses Softwareentwicklungsprozesses dienen können. Die Verfügbarkeit der Domänenbeschreibung und im Strukturierungsparadigma verankerte Analysemethoden ermöglichen eine langfristige Interpretierbarkeit der Modelle. Derartige Programme eignen sich zur Modellerstellung, Modellbewahrung und zum standardisierten Modellretrieval. Sie sind besonders interessant für einen Einsatz in frühen Phasen, um Entwurfsaufgaben systematisch erfassen zu können. CAD-Systemtools der Kategorien A bis C können über das Retrievalinterface in späteren Phasen diese Modelle nutzen. Der Vorteil besteht in der Entkopplung von Modell und Algorithmus - die Modelle enthalten nur deskriptive Elemente und keine Informationen zur Organisation eines Programmablaufs[6]. Solche Tools agieren als Modelleditoren oder '*Design Repositories*'.

[6] Dies scheint ein Widerspruch zur Softwaretechnologie, die mit dem Objektbegriff ja eine Kapselung von Objektdaten *und* -methoden anstrebt. Da der Code (Methoden), um den programmiersprachliche Objekte angereichert sind, i.allg. maschinenabhängig ist, kann er gerade *nicht* Gegenstand einer langandauernden Speicherung von Modelldaten sein! Dasselbe gilt für Objektdaten, die lediglich der Codeorganisation dienen.

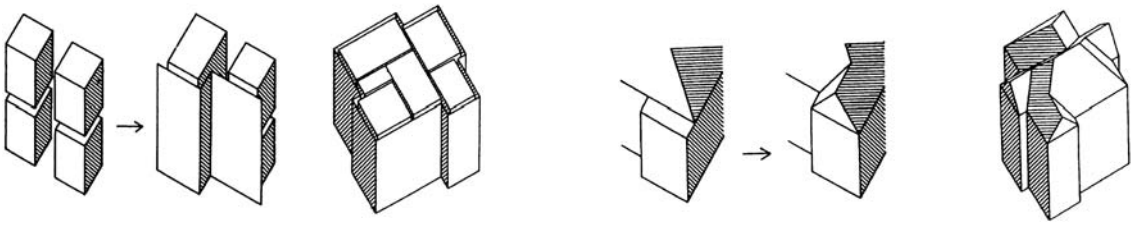


Abb. 2-6 a Regelbasierte Erzeugung der Kubatur von 'Queen Anne'-Häusern, (FLEMMING, [FLE90b])

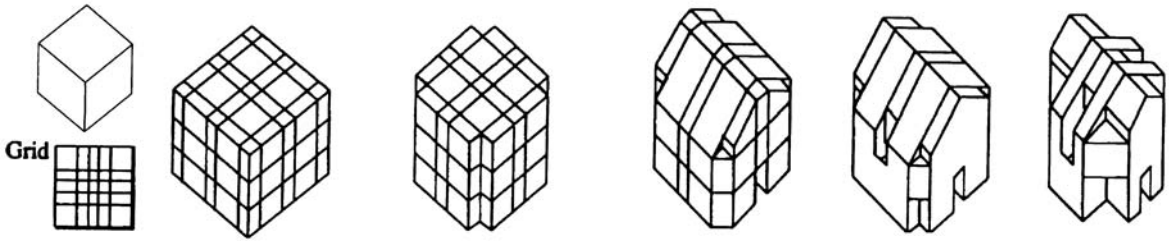


Abb. 2-6 b Generative Erzeugung von Kubaturen (SCHMITT, [SCH90])

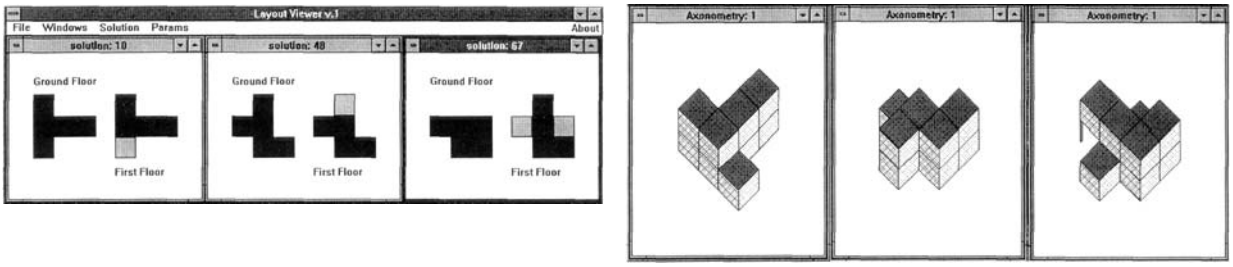


Abb. 2-6 c Generative Erzeugung einfacher Kubaturen (PETROVIC, [PET94])

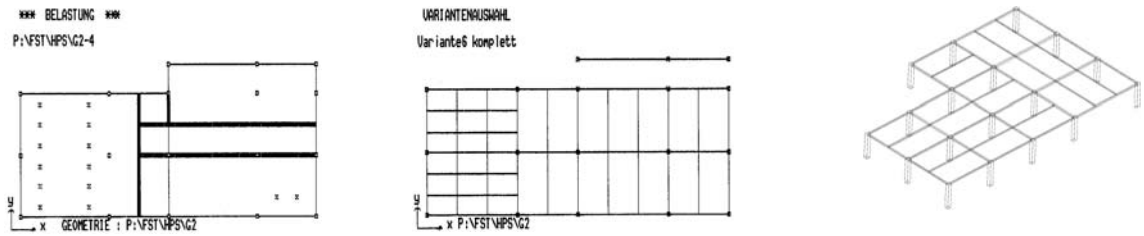


Abb. 2-6 d Generative Erzeugung von orthogonalen Trägerlagen (STEINMANN, [STE90])

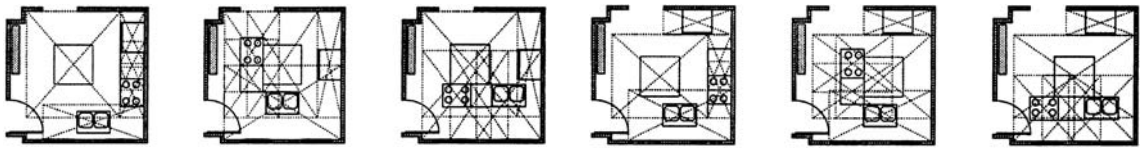
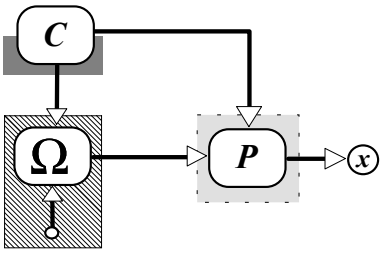
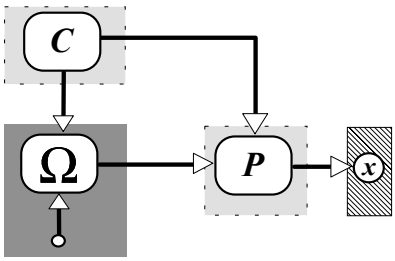


Abb. 2-7 Grundrißvariationen einer Küche durch das System LOOS (FLEMMING, [FLE90b])

(D) 'DESIGN REPOSITORY'

Tools der Kategorien B und C arbeiten (heute) über fixierte Domänenwelten, d.h. der Vorrat an Beschreibungselementen zum Erzeugen und Bewerten von Planungsobjekten ist beschränkt. Besonders für den (architektonischen) Vorentwurf bereitet das Erstellen statischer Produktmodelle Schwierigkeiten. Im Gegensatz zum 'Design Assistant', verfügt ein 'Design Repository' über kein fixiertes Produktmodell, sondern läßt dessen nutzerspezifische Erweiterung zu. Prozeduren zur Bewertung und Kontrolle treten weiter in den Hintergrund. Ein 'Design Repository' organisiert das simple Erzeugen, Editieren und Verwahren von Modellbeschreibungen. Anliegen ist das systematische Erfassen und Verwalten aller relevanten Projektinformation in allen ihren verschiedenen Dokumentationsformen. Der Aufbau und die Strukturierung von Bauwerksmodellen erfolgt in genau dem Abstraktionsgrad, der der jeweiligen Entwurfsphase angemessen ist.
Die Konzentration auf die Bauwerksbeschreibung ist die Grundlage der Stabilität dieser Modelle über den gesamten Gebäudelebenszyklus und damit ihrer durchgängigen Auswertbarkeit mittels CAD/ CAM/ CAFM[7]-Tools.
Gegenwärtig sind Systeme aus dem Bereich Facility Management am ehesten in diese Kategorie einordenbar (etwa der FM-Systemkern 'KOPERNICUS' [SCH95]). Einige Systeme aus dem Bereich der Künstlichen Intelligenz, soweit sie sich mit modellbasierter Konfiguration befassen, gehören ebenfalls in diesen Bereich. Für sie war jedoch nie eine direkte Nutzung durch den Anwender vorgesehen (Beispiel 'PLAKON' [CUN91]).

Modellgeneratoren 	• Aufbau des Objektmodells durch das System • implizites Kontextmodell zur Konsistenzprüfung des Objekmodells • implizites Bewertungsmodell von Entwurfsvarianten zum Einsatz in einer Suchstrategie bei der Objektmodellgenerierung • Ausdrucksmöglichkeiten im Rahmen der unterstützten Domäne stark eingeschränkt ⇨ *Einsatzbereich:* '*routine design'*
Modellvariatoren 	• Aufbau des Objektmodells durch den Nutzer • implizites Kontextmodell zur Konsistenzprüfung des Objekmodells • implizites Bewertungsmodell von Entwurfsvarianten zum Einsatz in einer Suchstrategie bei der Objektmodellvariation ⇨ *Einsatzbereich:* '*routine design*'

[7] Computer Aided Facility Management

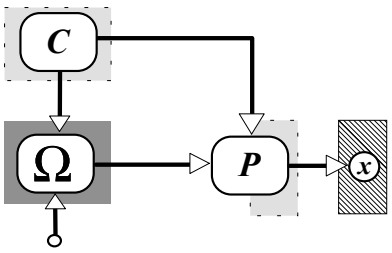
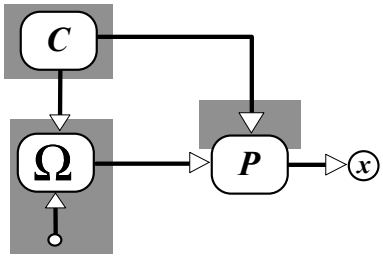
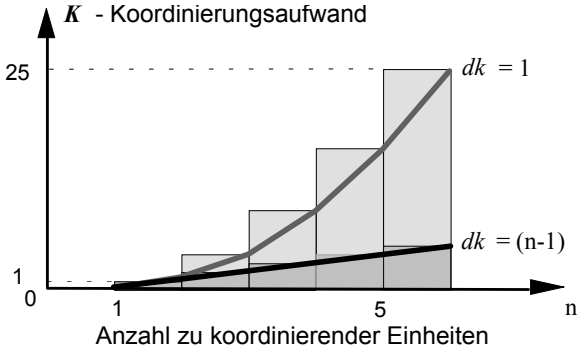
Modellevaluatoren 	• Aufbau des Objektmodells durch Nutzer • bedingter Aufbau oder implizites Kontextmodell zur Konsistenzprüfung des Objekmodells • bedingter Aufbau oder implizites Bewertungsmodell zur Bewertung einer Entwurfsvariante ⇨ *Einsatzbereich* : '*routine design*', '*innovative design*'
Modelleditoren 	• Aufbau des Objektmodells durch den Nutzer ohne Einschränkung der Ausdrucksmöglichkeiten innerhalb des Modellparadigmas • explizites Kontextmodell ohne automatisierte Konsistenzprüfung des Objekmodells • kein oder explizites Bewertungsmodell des erreichten Objektmodellzustandes ⇨ *Einsatzbereich* : '*innovative design*', '*creative design*'
implizit im System gegeben — explizit durch Nutzer anzugeben — durch System generiert	

Abb. 2-8 Klassen modellbasierter CAD-Systeme sowie deren Arbeitsfelder

2.3 Architektur durchgängiger CAD-Systeme im Bauwesen

Der aus dem Mittelalter bekannte omnipotente Baumeister, der alle Funktionen des Planens, Leitens und Überwachens der Bauwerkserstellung in Personalunion übernahm, ist heute ersetzt durch zusammenarbeitende Spezialisten. Nur so lassen sich komplexe Bauwerke in kurzer Zeit planen und herstellen. Die Zusammenarbeit innerhalb des Teams erfordert jedoch einen Zusatzaufwand für die erforderliche Koordination. Dieser Aufwand steigt überproportional mit der Gruppengröße, aber nur so wird die erforderliche Expertise verfügbar.



dk	Vernetzungsdichte
$1 \le dk \le (n-1)$	partiell vernetzt (realistisch)
$dk = 1$	vollständig vernetzt
$dk = (n-1)$	sequentielle 'Vernetzung'
$K = \frac{n(n-1)}{dk}$	Aufwand

Abb. 2-9 Aufwand beim Systemaufbau aus kommunizierenden Einheiten[8]

[8] im Softwareengineering ist der Term als Brooksches Gesetz zur Abschätzung des Kommunikationsaufwandes kooperierender Programmierer bekannt [DEN92]

Ein solches komplexes Team von Spezialisten zieht ein ebensolches System von spezialisierten CAD-Tools nach sich. Auch hier ist die Hoffnung auf ein monolithisches Programm zur Unterstützung aller Teilprozesse der Planung und Leitung völlig unrealistisch. Es stellt sich die Frage nach der Gestaltung durchgängiger, also koordinierter Systeme aus Einzeltools. Die Integrationsvariante von CAD-Tools wird in [GI89] untersucht und Vorschläge zu deren Realisierung abgeleitet ('CAD-Referenzmodell', siehe auch [ABE95]). Leider ist damit kein Konzept zur allgemeinen Büroautomatsierung, d.h. zur Integration aller mit einem Projekt assoziierten Dokumente verbunden.

Für die Integration von CAD-Tools werden hier zwei grundsätzliche Wege unterschieden :

(A) Modellaustausch ('Kopplung' [RAN92], 'äußere Integration' [GI89])

(B) Modellsharing ('Integration' [RAN92], 'innere Integration' [GI89])

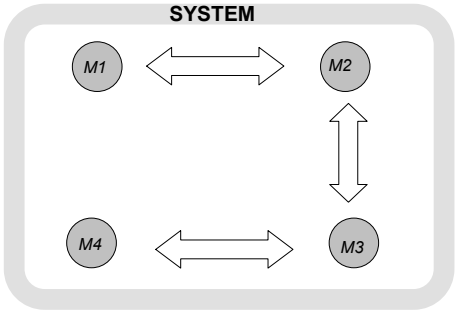
Beide Formen stellen Extreme dar, die so in der Praxis wohl nicht zu finden sind.
Als Mischform von A und B wird hier die

(C) Modellclusterung

vorgeschlagen. Die Abb. 2-10 a - d veranschaulichen für verschiedene Systemstrukturierungsvarianten den zu erwartenden Aufwand, der für die Erstellung und dauerhafte Aufrechterhaltung eines Systemverbundes zu erwarten ist. Die hier nachvollzogenen und erweiterten Betrachtungen aus [GI89], [KRE94] beziehen sich letztlich nur auf die Syntax der Schnittstellen und damit auf die prinzipielle Funktionsfähigkeit des Verbunds. Wenn z.B. ein völlig neues Bauelement in einer Entwurfssoftware erzeugbar wird, hat dies zur Folge, daß ein Modellupdate in allen Tools des Systems notwendig wird, nicht nur in den über Schnittstellen direkt verbundenen. Der Aufwand liegt also letztlich wieder bei dem von A2!
Abgeschätzt wird der Aufwand K, hier in Form der Anzahl zu implementierender Schnittstellen bzw. den zu leistenden Modellinterpretationen in Abhängigkeit von der Struktur des Systemverbundes. Unter der statistischen Annahme, daß an jeder Schnittstelle gleicher Verlust auftritt, wird der größte zu erwartende Informationsverlust $\max V$ zwischen zwei Einzeltools des Systems für jede Strukturierungsvariante überschlagen.

Systemstruktur A1

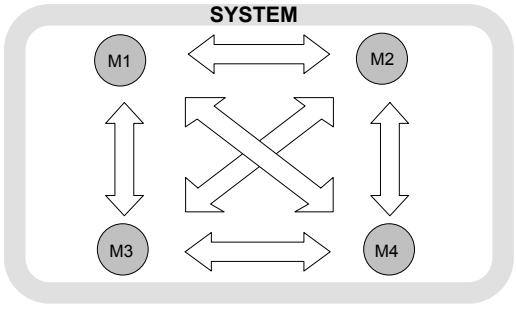
bei $dk = 1$ sequentielle Systemstruktur



$\max V = (n-1)$, $K = (n-1)$

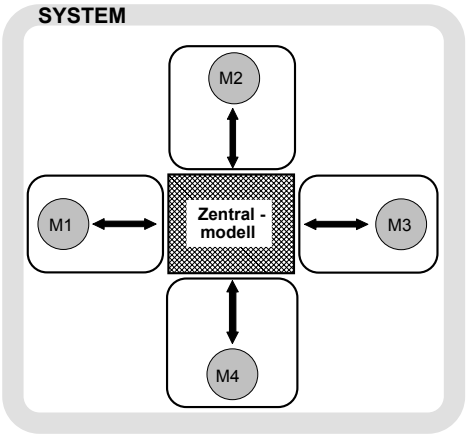
Systemstruktur A2

bei $dk = n$ vernetzte Systemstruktur



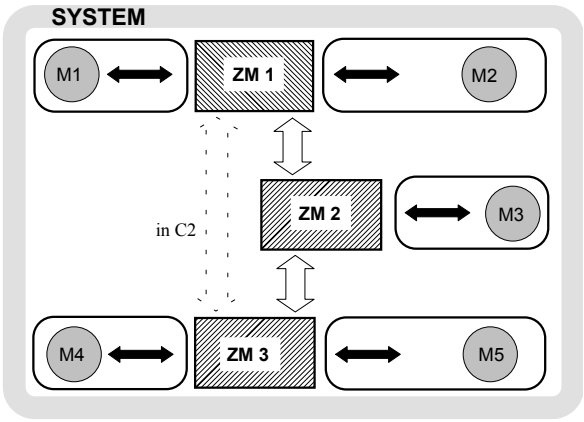
$\max V = 1$, $K = n(n-1)$

Systemstruktur B



$\max V = 2\,, \quad K = n$

Systemstruktur C (C1 und C2)



C1: Struktur wie A1 (sequentiell) für den Austausch zwischen den Clustern

$$\max V = 2 + (m-1)\,, \quad K = (m-1) + \sum_{i=1}^{m} n_i$$

C2: Struktur wie A2 (vollständig vernetzt) für den Austausch zwischen den Clustern

$$\max V = 3\,, \quad K = m(m-1) + \sum_{i=1}^{m} n_i$$

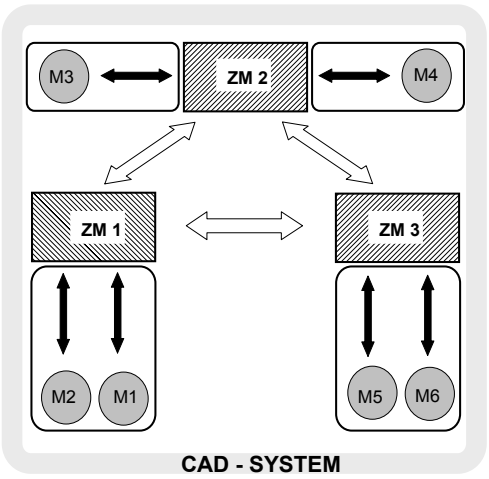
n - Modulanzahl
m - Anzahl der Cluster
n_i - Anzahl der dezentralen Modelle je Cluster
$\max V$ - max. Informationsverlust

Abb. 2-10 a - d Varianten der Systemstrukturierung

Verlustbehaftete Datenübertragung ist die Regel, nicht die Ausnahme. In [HAA88] ist zu lesen, daß selbst bei der Übertragung von reinen Zeichnungsdaten via IGES-Schnittstelle durchschnittlich nur 27% der IGES Funktionalität von 12 bekannten CAD-Systemen benutzt bzw. unterstützt wurde. Selbst einfache 2D-Geometriedaten wurden fehlerhaft übertragen, während der innere Aufbau der Zeichnungen, die Semantik von Schraffuren und Bemaßungen oftmals verloren gingen. Als ein weiteres Problem erweist es sich, daß es grundsätzlich nicht möglich ist, formal die Korrektheit von Schnittstellen nachzuweisen[9].

Damit ist verdeutlicht, worin das Problem gekoppelter CAD-Systeme besteht, wie das folgende eher optimistische Beispiel zeigen soll. Das System sei gemäß Variante **C2** (geclusterte Modelle, Cluster vollständig vernetzt) strukturiert.

[9] Das Problem ist aus dem Softwareengineering bekannt. Auch dort ist eine (automatische) Verifikation unmöglich, da eine formale Beschreibung von Programmsemantik benötigt würde.



Es gilt $n = 6$ Einzeltools, $m = 3$ Cluster
$n_i = 2$ für alle $i = 1,3$ Cluster

Für die Tools gilt, bedingt durch inhaltliche Weiterentwicklungen sowie durch Entwicklungen in Basissoftware und Hardware, eine Umschlagzeit von $t_t = 2$ Jahre für die 'major updates'. Für die zentralisierten Modelle sei eine Umschlagszeit von $t_m = 6$ Jahren veranschlagt. So ergeben sich für den Nutzer eines solchen Systems aus Einzeltools

$$\frac{1}{t_t} n + \frac{1}{t_m} \left(\sum_{i=1}^{m} n_i + m(m-1) \right) = 5$$ Updates pro Jahr (!)

wovon $\frac{1}{t_t} n + \frac{1}{t_m} m = 3.5$ inhaltlich begründet sind.

Abb. 2-11 Beispiel einer Systemstrukturierung

Daß die geforderte Updatefrequenz in der Praxis unrealistisch ist, ist offensichtlich. Die Qualität der Modellübertragung, z.B. von Modul M3 über 3 Schnittstellen nach Modul M5, wird hierbei nicht einmal betrachtet!
Ein weiteres Handicap für die Systemstrukturierung der Form A und entsprechend bei C ist, daß die inhaltliche Konsistenz von Datenbeständen innerhalb der Teilmodelle nicht automatisch zu sichern ist. Es gibt entlang von Schnittstellen keine Mechanismen, um beim Update eines Modellelements in einem Tool dessen Entsprechung in allen anderen betroffenen Tools zu 'informieren'[10].
Folgt man den Ansätzen B und C, so ist zum Erstellen der zentralen Produktmodelle ein Konsensbildungsprozeß von Nöten. Der könnte durch eine freiwillige Zusammenarbeit der Toolhersteller erfolgen. Diese Abstimmungsprozesse sind aber wesentlich langsamer als die Dynamik bei der inhaltlichen Entwicklung der Tools. Die Marketingpolitik führender CAD-Hersteller läßt hierzu auch keine ernsthafte Bereitschaft erkennen. Der andere Weg wäre ein staatlich sanktioniertes Modell, welches durch nationale und internationale Gremien zu standardisieren wäre. Vorteil wären die höhere Verbindlichkeit und Stabilität. Das Bundesbauministerium hat auf Anregung der Bauindustrie 1972 das 'Standardleistungsbuch' StLB entwickelt, ein nach Gewerken gegliedertes Codierungssystem zur Standardisierung von Ausschreibungstexten[11]. Andererseits hinken derartige Standardisierungen dem technischen Fortschritt bedenklich hinterher, da die entsprechenden Verfahren langwierig sind. Als Beispiel sei hier die STEP-Schnittstelle[12] angeführt (siehe u.a. [AND92]) :

- Entwicklungsbeginn war vor 1984 durch die ISO
- nach wie vor nicht verbindlich (z.T. auch gar nicht angestrebt)
- eine Fülle von Einzelnormen, die z.T. in sich inkonsistent sind (z.B. STEP-FEM, Umfang 110 Seiten)
- gegenwärtige Benutzung von STEP-2DBS (STEP *2D B*au*S*ubset, [HAA93]), aber es ist noch kein System am Markt, das einen Datenaustausch über diese Schnittstelle realisiert.

[10] neue Entwicklung wie verteilte Objekte, etwa in SOM / DSOM (von IBM) und die dort erhältlichen 'replication frameworks' bieten Ansätze, die auf eine Eignung für die CAD-Systementwicklung zu untersuchen sind.
[11] Erhältlich als 'Baudatenbank' der Firma Baudatenbank GmbH
[12] „Standard for the Exchange of Product Model Data"

Ob für CAD-Modellwelten eine Normung ähnlich der für Schraube/ Mutter-Verbindung gelingt, ist zweifelhaft und noch heute existieren mit dem Zoll- und dem metrischen System in Europa zwei parallele Standards. Im übrigen ist nicht jede Domäne des Bauwesen in gleicher Weise für eine Standardisierung geeignet. Kreative Phasen architektonischer Entwurfsprozesse mit ihrer starken Subjektivität (oder besser Individualität) entziehen sich einer Normung von vornherein.
Der eingeschlagene Weg der Standardisierung ist notwendig und sinnvoll, doch wird auf dieser Basis auch in absehbarer Zeit nicht mit durchgängigen CAD-Systemen zu rechnen sein. Es wird in jedem Tool eine zumindest partielle Wiedereingabe von Modelldaten, d.h. eine Remodellierung notwendig bleiben, mit allen Problemen der Modellkonsistenz, die dies nach sich zieht.

2.4 Anforderungen an die CAAD-Systementwicklung

Da die angeführten Probleme prinzipieller Natur oder zumindest gesellschaftlich determiniert sind, muß die Entwicklung von CAD-Tools und Systemen reagieren. Dies ist auf zwei Wegen möglich. Der eine zielt neben einer Forcierung der Standardisierung auf eine Verbesserung des Softwareentwicklungsprozesses mit den folgenden Zielen:

- allgemeine Verbesserung der Funktionssicherheit und Funktionalität
- Verkürzung der Reaktionszeiten bei induzierten Updates
- bessere Adaptierbarkeit sowie kunden- bzw problemspezifische Konfigurierbarkeit von Tools und Systemen[13]
- Entwicklung hochwertiger, d.h. semantisch reicher und adäquater Domänenmodelle
- Überführbarkeit externalisierter Modellwelten (z.B. im Rahmen einer objektorientierten Analysephase) in Programmcode, um den semantischen Abstand zwischen konzeptueller Ebene und Programmcode zu verringern.

Der andere Weg zielt auf einen anderen Umgang mit Modellen. Statt beim Modelltransfer- bzw. -sharing die korrekte Übertragung von Semantik zu erwarten, sollte diese Übertragung als ein inhaltlicher Interpretationsprozeß verstanden werden, der er ja tatsächlich ist. Die gegebene Subjektivität, Unvollständigkeit und Unschärfe von Modellen ist durch Programme zu tolerieren. Jedes Modell eines konkreten Bauwerks gehört in einen spezifischen Kontext, der, neben vielem anderen, auch die (möglicherweise standardisierte) Domänenmodellierung zum Zeitpunkt der Entwicklung der benutzten Programme beinhaltet. Bei der kreativen Lösung von Designaufgaben gehört jedoch das Schaffen dieses interpretatorischen Kontextes selbst zur Lösung, ist also durch den Architekten zu leisten [RIT92]. Er muß folglich über Möglichkeiten verfügen, diesen Kontext explizit zu machen. Benötigt wird dazu ein Metamodell, also ein Modellstrukturierungsparadigma zur Abbildung des Domänenmodells. Dieses Metamodell begründet die Interpretationsvorschrift für die Instanzen der Domänenmodelle, d.h. für die konkreten Bauwerksmodelle. Mit dem Paradigma der *Objektorientiertheit* ist ein mögliches Metamodell gegeben.

[13] RANGLACK formuliert: „CAD-Systeme beginnen beim Linken“, gemeint ist, daß ein Satz generischer CAD-Module in Form von Linklibraries vorliegen sollte, so daß mit Hilfe weniger speziell zu entwickelnder Module durch Linken spezifische CAD-Systeme generierbar wären.

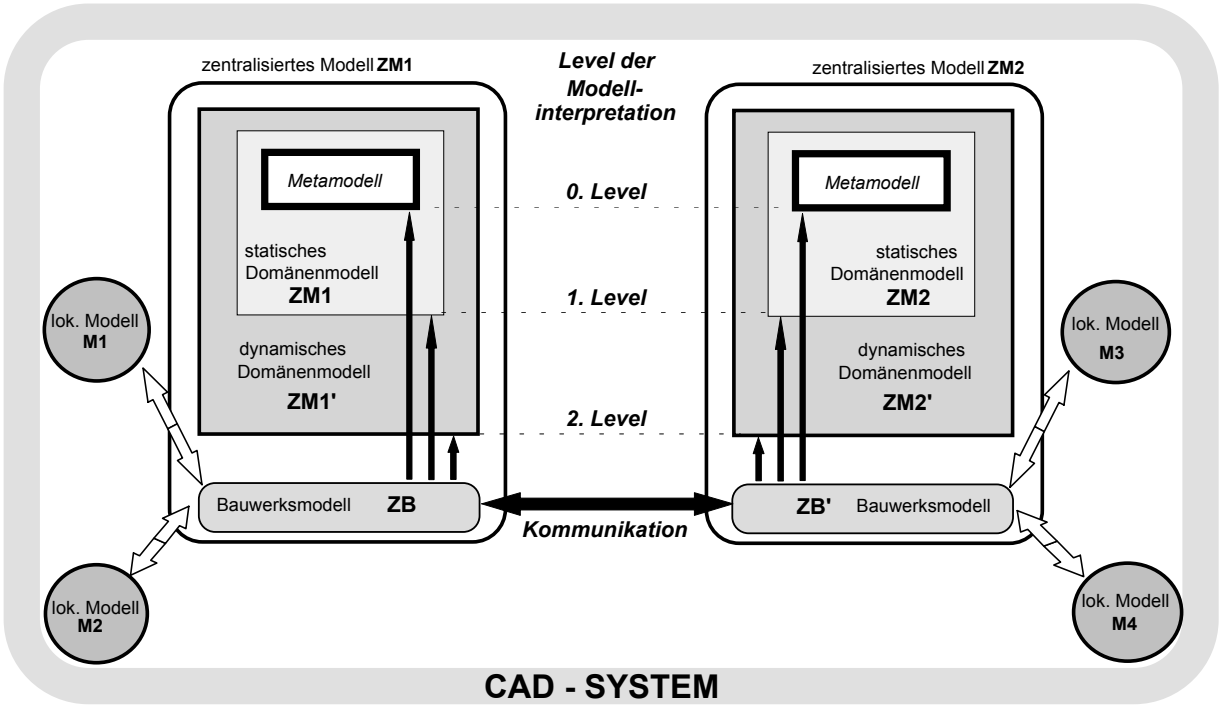


Abb. 2-12 Strukturierungsvorschlag für dynamische, modellbasierte CAD-Systeme

Auch wenn man dem ersten Weg, d.h. der Verbesserung der Qualität der Softwareentwicklung folgt, ist unstrittig, daß explizite Domänenmodelle erstellt werden müssen. Mit der EXPRESS-Norm [EXP89] existiert hierzu bereits eine standardisierte Notation.

Zusammenfassend sollen vier Feststellungen formuliert werden, an denen sich die weiteren Aussagen und Vorschläge zur Stützung früher Entwurfsphasen orientieren:

(1) Die dynamische Entwicklung der Computertechnik sowie der Bau- bzw. Planungstechnik machen eine ebenso dynamische Anpaßbarkeit, Erweiterbarkeit und Wartungsfähigkeit der Software nötig.

(2) CAD-Systeme werden problem- und nutzerspezifisch aus Tools konfiguriert. Deren Modellwelten sind nicht disjunkt, aber keinesfalls identisch.

(3) Domänenmodelle früher Entwurfsphasen sind nur bedingt konsensfähig, sie sollten daher dynamisch anpaßbar und erweiterbar sein. Als Grundlage eines Softwareentwicklungsprozesses eignet sich ein Metamodell, mittels dem Domänenmodelle explizit beschrieben werden können.

(4) Die Semantik konkreter Bauwerksmodelle ist an den Kontext der Domänenmodelle gebunden, auf den die Programme beruhen, mit denen sie erzeugt wurden. Zur fortgesetzten Interpretation von Bauwerksmodellen werden die Domänenmodelle explizit benötigt.

3. ASPEKTE DES ARCHITEKTONISCHEN ENTWERFENS

Ausgangspunkt jeder baulichen Veränderung unserer Umwelt ist ein gesellschaftlicher oder privater Bedarf, der dann einen Planungsprozeß initiiert. Die Planung, das Design eines Bauwerks und dessen Herstellung sind zweckgetrieben, um den aufgetretenen Bedarf zu befriedigen. Die Bauwerke müssen in diesem Sinne *funktionieren*. Für einen Zweck gibt es ein ganzes Spektrum von Möglichkeiten. Sie reichen von primär architektonischen Forderungen, wie das Erzielen einer künstlerischen 'Aussage' durch die gewählte Formensprache, simplen Anforderung aus der geplanten Nutzung wie '3000 m² Bürofläche', bis hin zu ingenieurtechnischen Bauwerken, die auf die Realisierung eines komplexen technischen Nutzungsprozesses zielen (siehe Trinkwasserbehälter [GYA94]).

In Anlehnung an JOEDICKE ([JOE76]) soll hier unter Architektur oder besser architektonischen Objekten folgendes verstanden werden:

> „ Architektur sind konstruierte, benutzbare und gestaltete Artefakte.
> Diese Artefakte nehmen in unserer Umwelt einen bedeutenden Raum ein."[1]

Hiermit sind im wesentlichen die Aspekte des architektonischen Schaffens benannt. Im Mittelpunkt steht das materielle Gut Bauwerk mit seiner raumgreifenden Wirkung. Dem Architekten kommen als Koordinator der Abgleich von *Funktion*, *Form* bzw. *Gestalt* und *Konstruktion* des Bauwerkes zu ([HAR80], [WEN90], [DON94]).

Gleichzeitig kennzeichnen diese Aspekte die möglichen Einstiegspunkte in den Entwurfsprozeß. Sie werden in JOEDICKE [JOE93] wie folgt beschrieben:

Funktionsorientierter Einstieg: Einige „werden mit der exakten Erfassung des Raumprogrammes als Einstieg in den Entwurfsprozeß beginnen. ... Es stellt sich dabei sofort die Frage nach der An- und Zuordnung der geforderten Flächen."

Formorientierter Einstieg: „Form ist ein entscheidender Faktor der Architektur und bestimmt deren Ausdruck; Form entsteht aber nicht von selbst, wenn der Grundriß in Ordnung ist, wie häufig behauptet wird. Entscheidend ist vielmehr, in welchem Stadium des Entwurfsprozesses Formfestlegungen notwendig und sinnvoll sind."

Konstruktionsorientierter Einstieg: Einige werden nach „einer klaren Struktur oder Konstruktion suchen, um von hier aus den Einstieg in den Entwurfsprozeß zu finden. Auch die Behandlung einer Konstruktion ist ... immer von Wertvorstellung abhängig." Nur in Sonderfällen, etwa „bei extremen Spannweiten, ... ‚bestimmen das statische System und das Material die Form."

Welcher Einstiegspunkt gewählt wird, hängt von der Art des formulierten Bedarfs, von der Komplexität des Entwurfsproblems und dessen Kontext, seinem Neuheitsgrad sowie von persönlichen 'Vorlieben' des Architekten ab.

Die Bedeutung des funktionsorientierten Einstiegs resultiert aus den Besonderheiten von Bauwerken als Entwurfsgegenstand. Deren Universalität als bauliche Hülle für die verschiedensten Nutzungstechnologien bedingt, daß im Gegensatz zu zweckgebundenen technischen Konstruktionen, eine explizite Analyse des (initialen) Bauwerkszwecks nötig ist.

---

[1] VITRUVIUS (römischer Architekt des 1. Jhr. v. Chr.) schrieb über Architektur [MSE94]:
„ein gutes Bauwerk hat drei Eigenschaften: Benutzbarkeit, Festigkeit und Ergötzlichkeit."

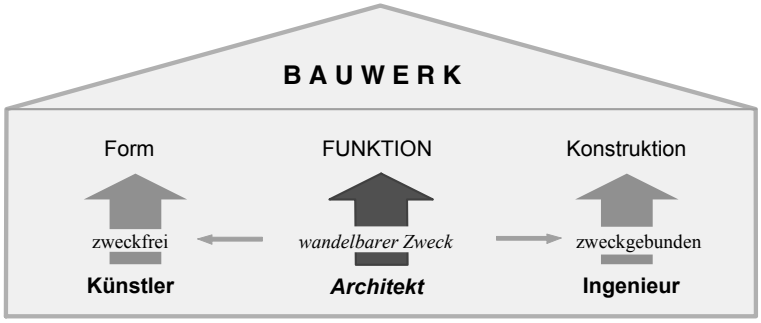


Abb. 3-1 Funktion als Mittler zwischen Form und Konstruktion des Bauwerks

Dies sollte sich auch in den angebotenen CA(A)D-Tools niederschlagen. Gegenwärtig ist bei modellbasierten CAAD-Systemen die Funktionalität der Entwurfselemente von vornherein festgelegt oder aber unbekannt. In diesen Systemen werden nur Wirkungsziele (Zweck) in die Auswertung einbezogen, die generisch für Bauwerke sind, wie z.B. seine Trägfähigkeit. Für den nutzungsspezifischen Zweck von Bauwerken existiert keine allgemeingültige Beschreibung.

3.1 Funktionaler Entwurf

Funktionaler Entwurf heißt Analyse der intendierten Benutzung im Rahmen vorhandener Ressourcen und eines lokalen Kontextes. Deren Ergebnis ist eine Spezifikation des Bauwerks. Spezifiziert werden dabei Elemente des gestalterischen oder konstruktiven Entwurfs. Restriktionen stammen darüber hinaus auch aus Bauvorschriften sowie grundlegenden technischen oder sozialen Gesetzmäßigkeiten, bis hin zu Spezifikationen, die das Einhalten einer gestalterischen Formensprache zum Ziel haben. Methodisch kann dies in Form einer Tätigkeitsanalyse (Abb. 3-6) erfolgen, wie sie bei SCHÖNFELD ([SCH92]) beschrieben wird (Abb. 3-2a). In '*Tätigkeitsdatenblätter*' (Abb. 3-2c) werden die Tätigkeiten

- identifiziert
- möglichst umfassend beschrieben (was, wann, warum, womit)
- mit Raumanforderungen versehen
- zugehörige Empfehlungen und Vorschriften identifiziert
- visuelle, physische, soziale und raumklimatische Anforderungen aufgestellt.

Das Ergebnis der Analyse wird

- in textueller Form als Raumprogramm,
- als grafischer Funktionsplan in Form eines '*bubble diagrams*' (Abb. 3-2c, 3-3, 3-5),
- als matrixartiger Funktionsplan (Abb.3-2b)

dargestellt. Im Analysevorgang selbst abstrahiert man von Geometrie, fordert aber die Existenz bestimmter Lage- und Erreichbarkeitsrelationen zwischen den Nutzungseinheiten. Damit wird eine Topologie beschrieben, die i.allg. zwischen Raumgruppen, Räumen oder Raumzonen realisiert werden soll. Die 'bubble diagrams' enthalten neben rein topologischen auch (wenige) geometrische Informationen, wie Fläche und Proportion der Nutzungselemente (dargestellt durch Größe und Form der Graphknoten) sowie deren Orientierung im Grundriß (Lage der Knoten). Die Beschreibung der Nutzungseinheiten erfolgt in textuellen, symbolischen und numerischen Daten. Für Funktionsmodelle liegt ein sehr hoher Abstraktionsgrad vor.

A) AUFLISTEN DER TÄTIGKEITEN

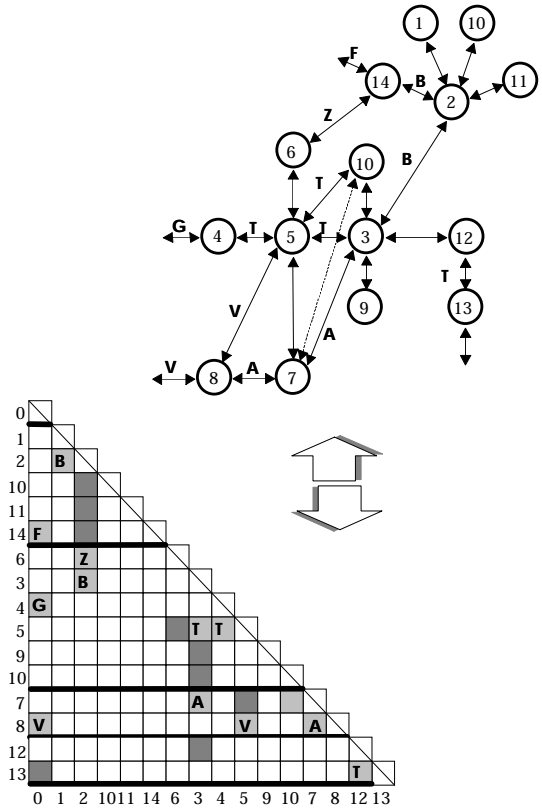
BEISPIEL:
HALLE FÜR TURNEN UND SPIELE

PRIMÄRTÄTIGKEITEN:
1 WARTEN
2 BEZAHLEN
3 UMKLEIDEN
4 GERÄTE AUFBEWAHREN
5 SPORT TREIBEN
6 DEN TURNENDEN ZUSCHAUEN
7 DIE TURNENDEN BEAUFSICHTIGEN
8 VERLETZTE BEHANDELN
9 SICH WASCHEN, DUSCHEN
10 WC
11 ERFRISCHUNG ZU SICH NEHMEN
12 AUSZENGERÄTE AUFBEWAHREN
13 AUSZENSPORT TREIBEN

SEKUNDÄRTÄTIGKEITEN:
14 PARKEN
15 REINIGEN
16 VER- UND ENTSORGEN
17 HEIZEN

PERSONEN, DINGE:
Z ZUSCHAUER
T TURNENDER
B BESUCHER
G GERÄT
V VERLETZTE
A AUFSICHT
F AUTOFAHRER

B) TÄTIGKEITSDIAGRAMM SPORTHALLE



C) TÄTIGKEITSDATEN

TÄTIGKEIT: VERLETZTE BEHANDELN

BESCHREIBUNG (WAS, WANN, WARUM)

TURNENDE, DIE SICH VERLETZT HABEN, WERDEN BEI LEICHTEN FÄLLEN VOM SPORTWART BEHANDELT - DANACH WEITERTURNEN ODER NACH HAUSE GEHEN - BEI SCHWEREN FÄLLEN WERDEN SIE DORT UNTERGEBRACHT BIS DER ARTZ KOMMT, ODER DURCH KRANKENWAGEN ABTRANSPORTIERT

PERSONEN: VERLETZTE, TURNAUFSICHT, ARZT

DINGE: LIEGE, TISCH, 2 STÜHLE, HANDWASCHBECKEN, MEDIKAMENTENSCHRANK, TRAGE, SPIEGEL
ANMERKUNG: LIEGE VON 3 SEITEN ZUGÄNGLICH

VORSCHRIFTEN UND RICHTLINIEN: DIN 18032
DIREKTER AUSGANG INS FREIE : . . .
KLIMATISCHE BEDINGUNGEN : . . .
BELICHTUNGSTECHNISCHE BEDINGUNGEN : . . .
SICHERHEITSBEDINGUNGEN : . . .
TECHNISCHE EINRICHTUNGEN : . . .
ANFORDERUNGEN AN BAUTEILE : . . .

D) RAUMZUORDNUNGSSCHEMA:

SPORTHALLE 27,00 * 45,00 m
TEILBAR IN DREI ÜBUNGSEINHEITEN

ZONE FÜR GERÄTE-, LEHRRÄUME, MAGAZIN UND REGIERAUM - TURNSCHUHGANG

UK UMKLEIDEZONE
S SANITÄRZONE / STIEFELGANG

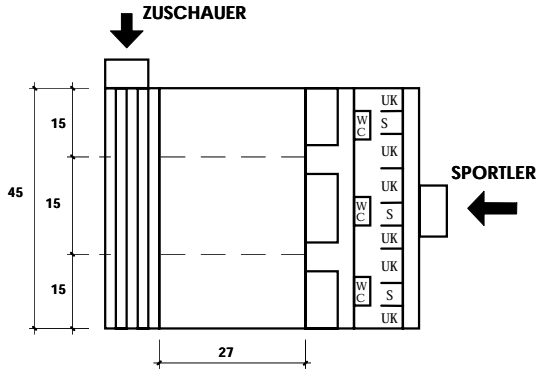


Abb. 3-2 a bis d Liste der Tätigkeiten, Tätigkeitsdatenblatt, Tätigkeitsdiagramm, Raumzuordnungsschema für eine 3-Felder Sporthalle (SCHÖNFELDER [SCH92])

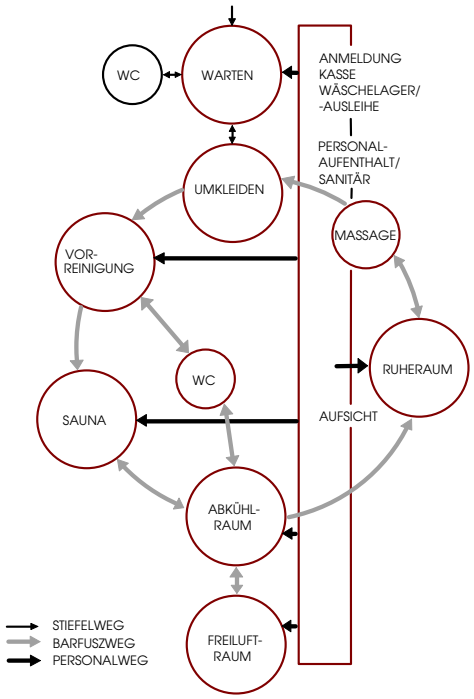


Abb. 3-3 Funktionskreis Sauna, allg. Forderungen nach HÖCKERT [HÖC76]

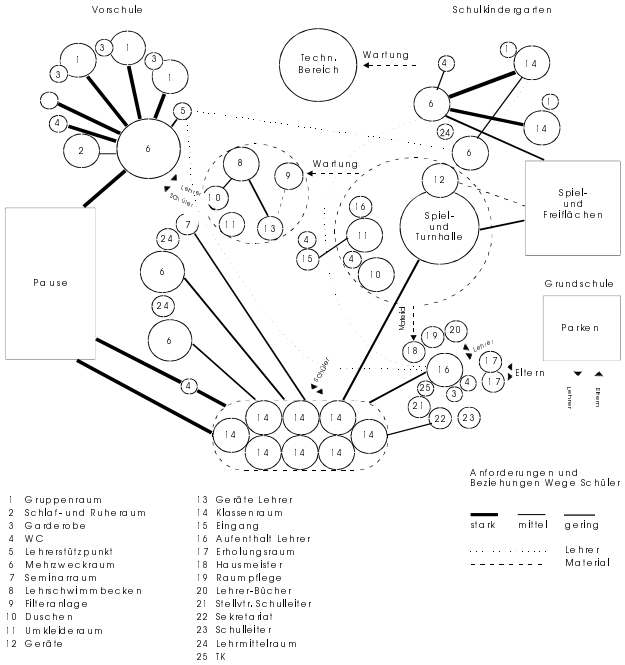


Abb. 3-4 Flächen- und Zuordnungsschema nach JOEDICKE [JOE76]

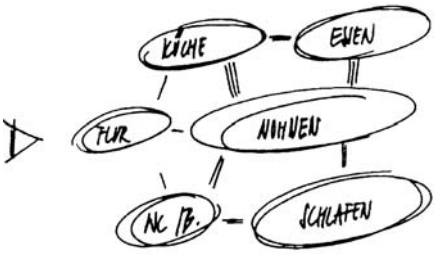


Abb. 3-5 Funktionsplan nach ERVIN aus [LIE93]
(vergl. zugehörige Form in Abb. 3-12 und Konstruktion in Abb. 3-20)

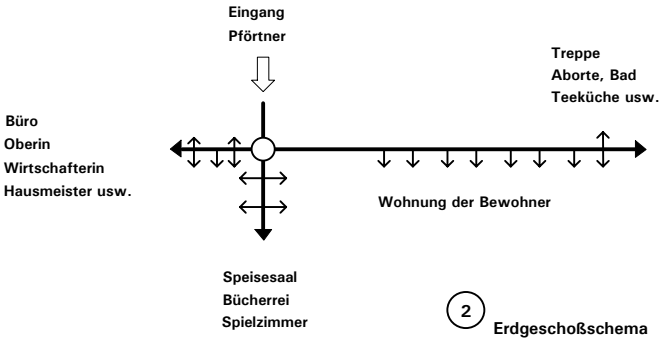


Abb. 3-6 nach NEUFERT [NEU51]
(vergl. zugehörige Form in Abb. 3-11)

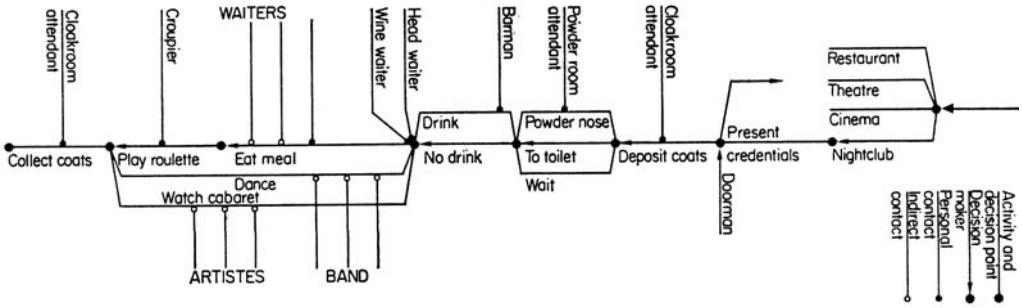


Abb. 3-7 Funktionsanalyse als Vorgangsgraph (nach BROADBENT [BRO73])

Die konstatierten Forderungen müssen nur 'ungefähr' gelten, die Beschreibungen sind 'grob' und häufig unvollständig. Die Nutzungseinheiten lassen sich als strukturelle Objekte in der späteren Konstruktion oft nicht mehr identifizieren. Im Rahmen eines CAD-Prozesses werden abstrakte Modellelemente wie das '*HAUS'* als Ganzes, Etagen oder Blöcke konstituiert, die den Kernbereich eines hierarchischen Bauwerksmodells bilden (siehe Aggregation Kapitel 6). Diese abstrakten, komplexen Elemente wiederum sind in der Phase der Bauwerksnutzung ('Facility Management') Gegenstand von Modellabfragen.

Obwohl die funktionale Analyse Teil der anrechenbaren Leistungen des Architekten (lt. HOAI Phase1) ist, existieren hierfür keine genormten Präsentationsformen, entsprechende Pläne sind nicht Teil der Projektunterlagen. Raumbücher, die noch am ehesten den Anspruch eines Raumprogrammes erfüllen, werden heute erst *nach* Erstellung der Grundrisse aufgestellt (so auch im System 'PH-CAD' [HEC94]). Das hat zur Folge, daß zwar häufig derartige Pläne erstellt werden, aber niemand sich die Mühe, macht diese Pläne sauber aufzuarbeiten und aufzubewahren. Alle Informationen dieser Analyse gehen verloren, sowohl für das Projekt selbst, als auch für einen Wissenserwerbsprozeß im Rahmen einer CAD-Tool Erstellung.
Bedingt durch die fehlende Dokumentationspflicht und damit Nachfrage besteht auch kein Angebot seitens der CAD-Systemhersteller zur Unterstützung des Funktionalen Entwurfs, wie eingehende Befragungen anläßlich der Messen 'Architekten, Computer, Software' 1994 und 1995 zeigten.

3.2 Gestaltorientierter Entwurf

Die Frage nach der Gestalt ist etwas, daß die Architekturtheorie seit langem beschäftigt. Sie faßt den Begriff allerdings sehr weit, bis hin zu einer kognitiven Betrachtung zur visuellen Perzeption, wie es die 'Gestaltbewegung'[2] tut. *Gestalt* ist mehr als *Form*, sie bezieht Form, Farbe, Textur, Akustik bis hin zu haptischen Eigenschaften ein. Es existiert eine Fülle von Literatur zu Fragen der Ästhetik, Proportion, Formensprache bestimmter Architekturströmungen.
Im Rahmen dieser Arbeit soll jedoch unter dem *'gestaltorientierten'* Entwurf das Finden der Kubatur eines Bauwerks verstanden werden. Dieser Einstieg in den Entwurfsprozeß betont die kreativ - skulptorale Arbeitsweise des Architekten[3]. Der Vorgang der Gestaltfindung befaßt sich mit Erzeugung, Benennung, Positionierung, Beschreibung und Komposition geometrischer Objekte, ist also an Formelementen orientiert. Deren Beschreibung erfolgt vorwiegend numerisch, schließt aber im Sinne des *Feature*-Ansatzes beliebige Annotationen nicht aus. Die Repräsentationsformen reichen von sehr abstrakten, freien Skizzen in einer privaten 'Handschrift' (Abb. 3-8, 3-12, 3-14), bis hin zu modernen Präsentationsformen mit fotorealistischen Bildern (Abb. 3-16), in 'Virtual Reality'-Umgebungen (Abb. 3-15) oder in Form von Animationen.

[2] Die 'Gestaltbewegung' [ROW87]: Diese Richtung kognitiver Psychologie erbrachte viele Beiträge zum Verständnis menschlichen Denkens, Problemlösungsverhaltens und visueller Perzeption (KÖHLER 1929, KOFFKA 1935, WERTHEIMER 1945). Das Konzept der *Gestalt* ist geprägt von der Idee einer holistischen Organisation von Informationen, welche den diskret mechanistischen Standpunkt des Assoziativismus ersetzt.
[3] Hierzu ADORNO, 1968, in [JOE76]: „... selbst die reinsten Zweckformen zehren von Vorstellungen wie der formalen Durchlässigkeit und Faßlichkeit, die aus künstlerischer Erfahrung stammen; keine Form ist gänzlich aus ihrem Zweck geschöpft"

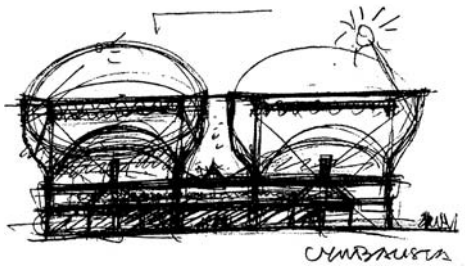


Abb. 3-8 Skizzenhafter Entwurf zur Cymbalista Synagoge Tel Aviv von BOTTA [HOL96]

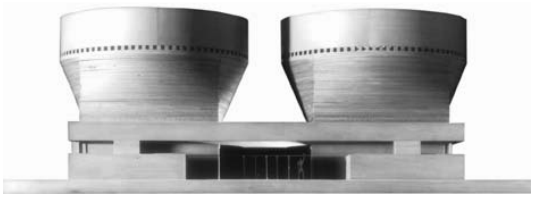


Abb. 3-9 Habtisches Modell der Cymbalista Synagoge Tel-Aviv von BOTTA ([NET1])

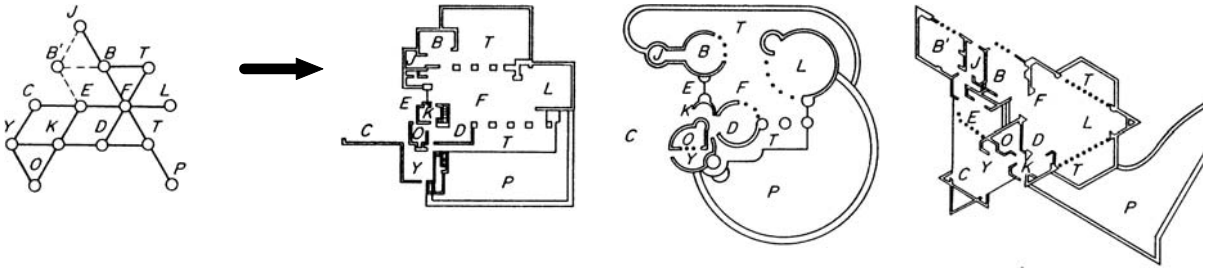


Abb. 3-10 Grundrißvarianten bei gleichem Funktionsgraphen nach WRIGHT (in [BRO73])

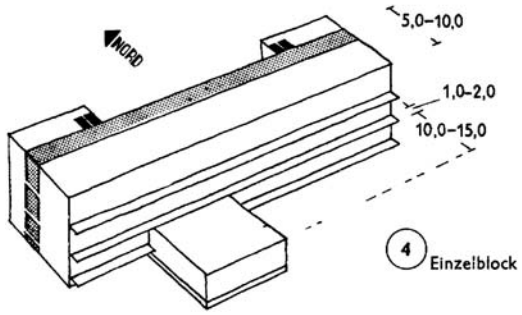


Abb. 3-11 Skizze eines Altersheims nach NEUFERT [NEU51]

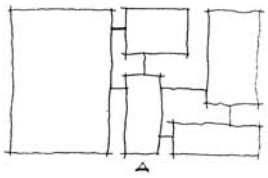


Abb. 3-12 Grundrißentwurf nach ERVIN (aus ([LIE93]), vergleiche Funktionsgraph Abb. 3-5 und konstruierter Grundriß Abb. 3-20)

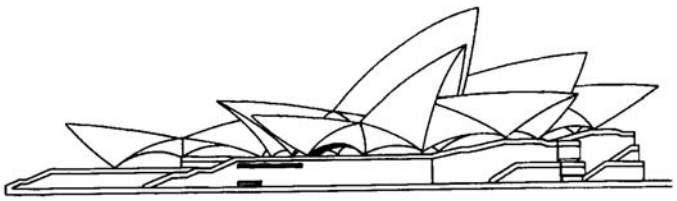


Abb. 3-13 Aufriß des Opernhauses Sydney nach UTZON (in [ROW87])

Abb. 3-14 frühe Skizze der “Einsteinturm“ nach MENDELSON (in [HOV94])

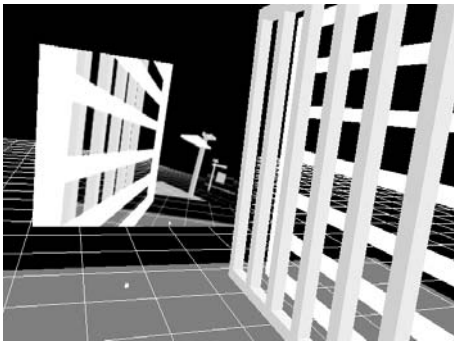


Abb. 3-15 VR-Entwurfsumgebung (REGENBRECHT)

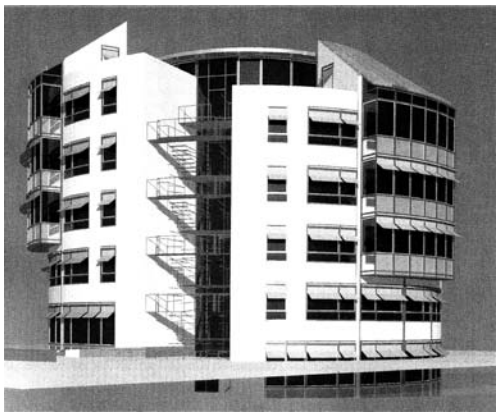


Abb.3-16 Gerendertes Gebäudemodell ([EBE92])

Die Darstellungsarten sind zwar ebenfalls nicht (gesetzlich) genormt, mit Ansichten und Projektionen (Abb. 3-11) liegen aber Darstellungsarten vor, für deren Layout eine breiter Konsens besteht. Der pure geometrische Entwurf unterliegt kaum Restriktionen. Bestenfalls Ästhetik und fundamentale Gesetze der Physik wirken einschränkend. Erst die Koordination mit dem *konstruktiven Aspekt*, d.h. die Überprüfung der Baubarkeit, wirkt restriktiv.
Obwohl dieser Entwurfseinstieg durch die Vielfalt der verfügbaren Geometriemodellierer auf den ersten Blick gut unterstützt scheint, ist dies bei genauerer Betrachtung nicht der Fall. Form*findung* mittels geometrischer Objekte, die dem Entwurfsstadium angemessen sind, findet nicht statt. Die Benutzung von vorwiegend mathematisch-geometrisch geprägten Universaltools (wie etwa AutoCAD AME und darauf basierenden Applikationen) bedingen die häufig beklagte 'Pedanterie des CAD Handlings' ([GI91]).
Die Formfindung ist ein traditionsreiches Feld der CAAD-Forschung, die im wesentlichen drei Hauptrichtungen verfolgt:

- Generierung von geometrischen Objekten (siehe 'design automats')
- weitere Verbesserung realitätsnaher Darstellungen, um zukünftiges Raumempfinden möglichst realistisch erlebbar zu machen (zu simulieren), wie Raytracing Verfahren (Übersicht in [WOS91]), licht- und schallenergetische Verfahren, VR-Simulationen
- Verfahren und Modelle für die frühen Phasen der Formfindung. Hierzu zählen Versuche
  - sprachliche Beschreibungsmittel auf linguistischer Basis anzubieten ([DAH96])
  - einfache, sehr effiziente Erzeugung von Basisgeometrien in immersiven 'Virtual Reality'-Welten ([DON95])
  - Geometriemodelle um Elemente von Fuzziness anzureichern ([RED95a])
  - einfache 3D-Welten aus 2D-Skizzen zu erstellen ('grobes CAD' in [KUN93]).

3.3 Konstruktiver Entwurf

Seine 'Zwitterstellung' zwischen Künstler und Ingenieur bezieht der Architekt letztlich auf Grund der Tatsache, daß seine Artefakte in einem aufwendigen technischen Prozeß realisiert werden und weitreichenden gesetzlichen Vorschriften genügen müssen – im Sinne von BEITZ ([BEI85]) könnte man ihn als 'kreativen Ingenieur' bezeichnen. Der konstruktive Aspekt ist als Einstieg in den Entwurfsprozeß eher selten. Er wird bei betont technischen Bauwerken (z.B. Wassertürme [GYA94]) gewählt oder wenn bestimmte Nutzeranforderungen

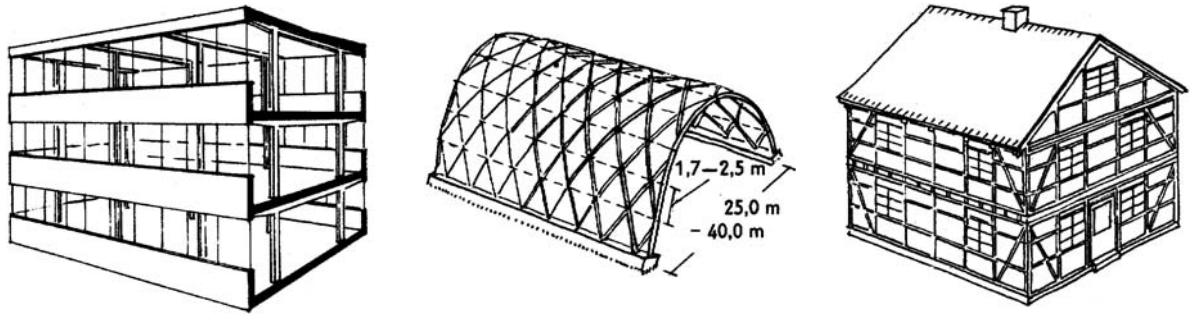
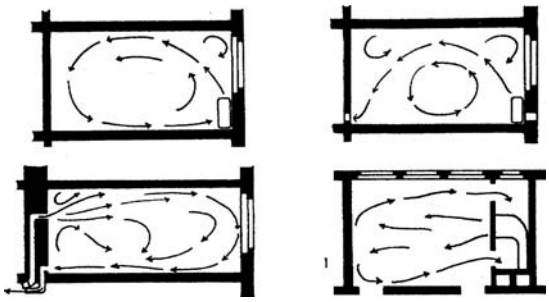
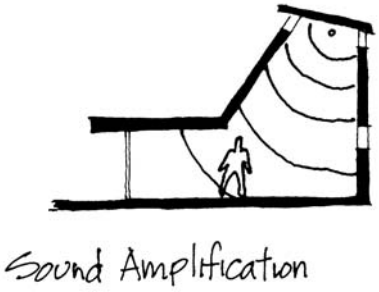


Abb. 3-17 a - c Konstruktionsfromen als gestaltbestimmendes Kriterium
(nach NEUFERT [NEU51])



nach NEUFERT [NEU51]



nach LASEAU [LAS89]

Abb. 3-18, Abb. 3-19 Bauphysikalisch Randbedingungen des konstruktiven Entwurfs

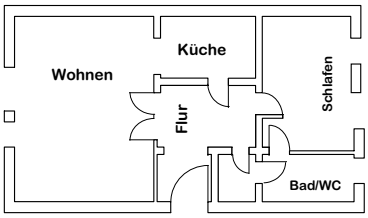


Abb.3-20 Grundrißvariante nach ERVIN aus [LIE93]
(vergl. zugehörige Form in Abb. 3-12 und Funktionsplan in Abb. 3-5)

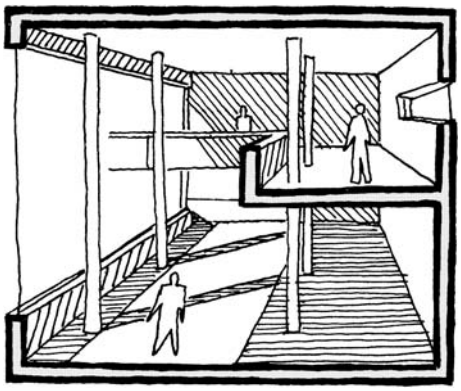


Abb. 3-21 Tragwerk und Innenraum
(nach LASEAU [LAS89])

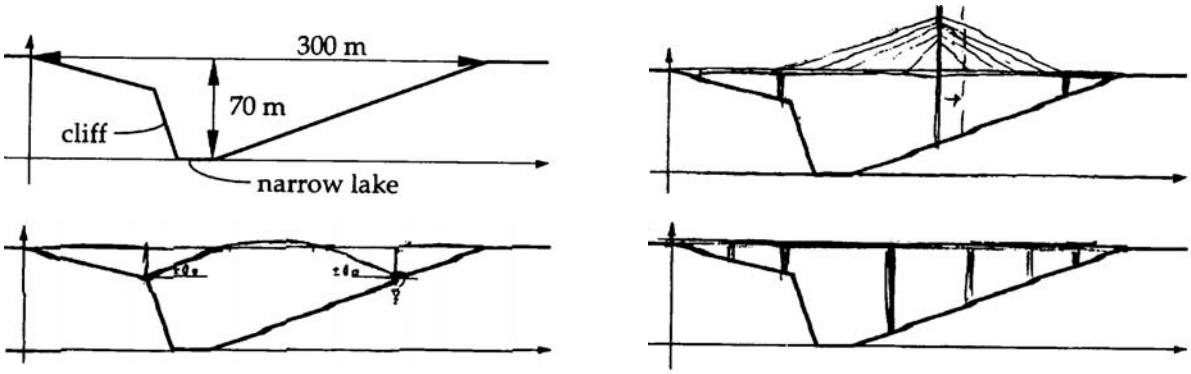


Abb. 3-22 Tragwerksentwurf als formbestimmendes Element im Ingenieurbau
(nach BERGAMO [BON95])

technische Leistungen erfordern, die über das 'Normalmaß' des technisch Machbaren hinausgehen (z.B. extreme Spannweiten). Vom Architekten wird das *Finden* einer Grobkonstruktion erwartet, nicht deren *Erfindung*. Die Selektion und Bewertung bekannter Prinzipkonstruktionen erfolgt unter den Gesichtspunkten Statik, Herstellungstechnologie, Bauphysik, Ökonomie und Ökologie. Die Modellelemente, die dieser Betrachtung unterliegen, sind *Bauteile* und deren stofflich-geometrische Eigenschaften. Sie werden i.allg. mit stark strukturierten numerischen Daten beschrieben sowie einem komplexen Netz von Relationen. Für das 'Endprodukt' dieser Überlegungen gibt es genormte Darstellungsarten in Form prüffähiger Unterlagen und Pläne. Diese Prüffähigkeit bezieht sich zum einen auf geforderte Leistungen, zum anderen auf die Einhaltung gesetzlicher Vorschriften wie etwa DIN-Normen. Die inneren Restriktionen eines solchen konstruktiven Modells ergeben sich aus technischen, technologischen und physikalischen Gesetzmäßigkeiten.

In frühen Phasen sind Tragwerksskizzen mit differenziertem Abstraktionsniveau gängiges Ausdrucksmittel (Abb. 3-17 bis 3-22). Auswertbare, berechenbare Modelle besitzen jedoch einen niedrigen Abstraktionsgrad. Hier, d.h. bei der Bewertung, wird Software sehr erfolgreich eingesetzt. Im Rahmen des Vorentwurfs besteht jedoch der Wunsch, auch unscharfe bzw. abstrakte konstruktive Konzepte einer Auswertung zu unterziehen, was die verfügbare Software jedoch kaum erlaubt. Die heutige Software betont den ingenieurmäßigen Entwurf und ist den Bedürfnissen des Architekten nicht angemessen. So beschreibt ROTTKE ([ROT92]) die unterschiedlichen oder doch zumindest anderswertigen Bedürfnisse von Architekten im Rahmen des konstruktiven Entwurfs und die daraus folgenden Anforderungen an CAAD-Tools wie folgt:

• *der Ingenieur rechnet – der Architekt zeichnet*	⇨ Kommunikation Programm-Architekt möglichst grafisch, numerische Ergebnisse nur bei Bedarf
• *der Ingenieur hat genaue Angaben – der Architekt ungefähre Vorstellungen*	⇨ Eingaben müssen auf das notwendigste reduziert werden, Teilkomponenten sollten überschläglich angenommen werden oder bleiben unberücksichtigt
• *der Ingenieur betrachtet statische Elemente – der Architekt Konstruktion oder Funktion eines Bauwerks*	⇨ Programme sollten der Denkweise des Architekten angepaßt sein
• *der Ingenieur benötigt exakte Ergebnisse – der Architekt benötigt überschlägliche Ergebnisse, diese aber schnell*	⇨ Vereinfachte Berechnungs- und Überschlagsverfahren
• *der Ingenieur benötigt Endergebnisse für die Dimensionierung – der Architekt benötigt Bewertungskriterien für den Entwurf*	⇨ Zwischenergebnisse der Tragwerksanalyse sollten grafisch, verständlich und nachvollziehbar dargestellt werden

- *der Ingenieur berechnet ein festgelegtes Tragsystem*
  – Architekt muß Alternativen konzipieren und bewerten
  ⇨ Varianten müssen mit geringem Aufwand erstellt werden können, Analyseergebnisse für Alternativen müssen vergleichbar dargestellt werden

Die Lösung konstruktiver Problemstellungen wird durch den Architekten i.allg. in einer Art von 'routine design' bearbeitet. Er verfügt über Kenntnisse von Prinzipkonstruktionen und ein breites Fallwissen, das auf den konkreten Fall angewendet wird. Es besteht deshalb die Hoffnung, daß fallbasierte Lösungsverfahren, wie sie u.a. im FABEL-Projekt[4] für den architektonischen Entwurf untersucht werden, den Architekten spürbar entlasten können. Eine Unterstützung des Entwurfs von Treppen wird in [OTT95] vorgestellt. Kreativitätsfördernd wirken diese Unterstützungen des Routineentwurfs durch die Verkürzung der Iterationszeiten bei der Bewertung von Varianten.

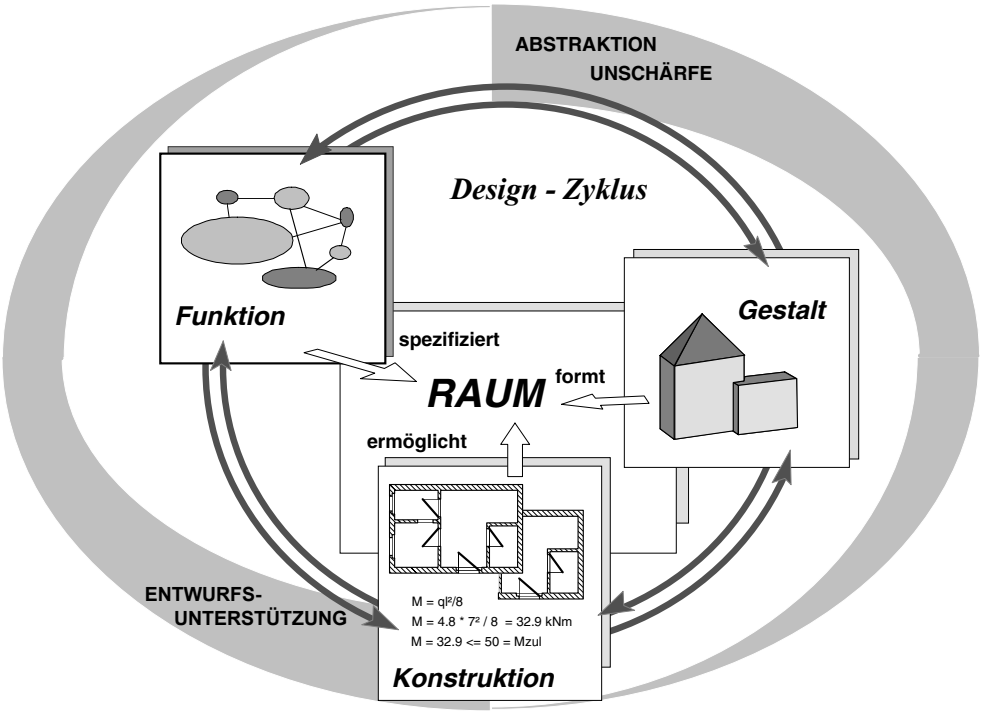


Abb. 3-23 Aspekte des Architekturentwurfs

Gegenwärtig wird der formorientierte Einstieg in den Entwurfsprozeß betont, was auch dem Image des Architekten als Künstler entgegenkommt. Dies schlägt sich (besonders) in der Ausbildung von Architekten nieder. Konstruktive Grundlagen sind kaum Gegenstand der Lehre, wie auch der Tragwerksentwurf lediglich 6 bis 10% ([ROT92]) des Gesamtpensums umfaßt. Dies ist aus historischem Blickwinkel um so verwunderlicher, entstand doch der Beruf des Ingenieurs erst Mitte des letzten Jahrhunderts, also lange nach dem des Architekten. Diese Fixierung auf die Form wird verstärkt durch die gegenwärtige Software, die stark geometrieorientiert ist. Der Grund ist das Vorhandensein geschlossener Theorien, wie der

[4] 'FABEL' ist ein Gemeinschaftsprojekt unter Führung der GMD mit dem Ziel, fallbasierte Techniken exemplarisch bei der konstruktiven -, Installations- und Ausstattungsplanung hochinstallierter Gebäude des MIDI-Systems (HALLER) zu erproben. Hierzu existiert eine ganze Reportserie, von der beispielhaft nur auf [BAK94] verwiesen sei.

'analytischen Geometrie', 'Differentialgeometrien' oder allgemeiner 'Computer Geometrien'. Diese Theorien sind gut eingeführt und vielen Nutzern bekannt. Für frühe systematische Entwurfsschritte und für spätere Facility-Management Prozesse gewinnt der Entwurf, der sich abstrakter Funktionselemente bedient (wie Räume, Flächen, Etagen) stark an Bedeutung. Tabelle 3-1 stellt noch einmal kurz die Einstiegspunkte in den Entwurfsprozeß gegenüber.

Funktion	***Gestalt***	***Konstruktion***
Funktionale Spezifikation des Bauwerks	Finden der Kubatur des Bauwerks	Finden einer Prinzipkonstruktion des Bauwerks
Analysierend-recherchierende Tätigkeit	Kreativ - plastische Tätigkeit	Prüfend-berechnende Tätigkeit
Entwurf bezogen auf Topologie, Struktur orientiert (Raum/Bereich)	Entwurf bezogen auf Geometrie, Formelement orientiert	Entwurf bezogen auf Statik und Technologie, Bauteil/ Baustoff orientiert
Beschreibung in symbolischen und numerischen Daten sowie in Relationen	Beschreibung vorwiegend numerisch, aber auch topologische Relationen	Beschreibung in stark strukturierten numerischen Daten und in Relationen
Hoher Abstraktionsgrad sowie Unschärfe und Unvollständigkeit	Differenzierter Abstraktionsgrad, Unschärfe bei Positionierung/Dimensionierung erwünscht	Niedriger Abstraktionsgrad, geringere, (aber) vorhandene Unschärfe, Variantenbildung
Restriktionen durch Gesetze, Vorgaben und Kontext	Restriktionen durch Physik und Ästhetik	Restriktionen durch techn. Wirkprinzipien, Physik und staatl. Gesetze
Keine Dokumentationspflicht für Funktionspläne	Bedingte Dokumentationspflicht	Umfassende Dokumentationspflicht
Keine genormte Repräsentation (Bsp.: Raumprogramme, Blasendiagramme)	Keine genormte Repräsentation (Skizzen, Projektionen, fototreal. Darstellungen, Modelle)	Weitgehend genormte Repräsentation in grafischer und textueller Form
Gegenwärtig hierfür keine Werkzeuge	Geometrisch-zeichnerische Universaltools, Unterstützung wenig architektengerecht	Gute Unterstützung später Phasen, aber unscharfe oder unvollständige Daten sowie Annahmen nicht unterstützt

Tab. 3-1 Besonderheiten der architektonischen Entwurfsaspekte

In der Praxis sind die genannten Entwurfsaspekete nicht trennbar. Wenn JOEDICKE von „Einstiegspunkt in den Entwurfsprozeß“ spricht, so meint er, daß der jeweilige Aspekt vorherrschend ist, tatsächlich wird, wenn auch mit unterschiedlicher Wertigkeit, gleichzeitig in allen Aspekten gearbeitet. Beim Architekten existieren darüber hinaus intensive Assoziation zwischen den Elementen der Entwurfsaspekte – befragt man ihn, mit welchem Elementetyp er gerade befaßt ist, so wird er nicht immer präzise Auskunft geben können.

Aus wissenschaftlich-technischer Sicht scheint der funktionsorientierte Aspekt für die frühen Entwurfsphasen von großem Interesse. Da die geplante Nutzung großen Einfluß auf eine an sich universelle bauliche Hülle hat, ist eine funktionale Spezifikation unumgänglich, wenn sie auch nicht immer explizit ausgeführt wird. Sie ist deshalb häufig „Einstiegspunkt“ in den Entwurfsprozeß, insbesondere für Bauwerke, die sehr komplex sind oder von ihrer Nutzungsfunktion dominiert werden. Exemplarisch sei hier auf Arbeiten von KRAUSE zur funktionalen Gestaltung im Krankenhausbau verwiese ([KRA83]).

Für die computergestützte Bewertung von Entwürfen sind gegenwärtig nur physikalische-technische oder gesetzliche Randbedingungen nutzbar. Ob eine Entwurf ‘funktioniert’, kann eine Software nur entscheiden, wenn eine explizite Spezifikation vorliegt. Sie schafft so die Voraussetzung für eine methodische Arbeitsweise. Der funktionale Entwurf wird gegenwärtig durch Software nicht unterstützt, obwohl in der entsprechenden Entwurfsphase Entscheidungen von großer ökonomischer Tragweite getroffen werden. So arbeitet etwa die Investvorbereitung i. allg. mit abstrakten Nutzungseinheiten. Die mangelnde Computerstützung ist um so verwunderlicher, da sie vergleichsweise einfach realisierbar scheint. Die Repräsentation von Funktionsgraphen ist kein technisches Problem und die erwartete Berechnungsfunktionalität in den Modellen beschränkt sich auf einfache kaufmännische Auswertungen sowie Erreichbarkeitsprobleme im Graphen.

Als Schlußfolgerungen werden im folgenden noch einmal kurz die Anforderungen an eine CAAD-Stützung in konzipierenden Entwurfsphasen im allgemeinen und des funktionalen Entwurfs im besonderen aufgeführt.

(1) Für die Konzeption komplexer Bauwerke besitzt der funktionale Entwurf große Bedeutung. Durch die Spezifikation wesentlicher Bauwerkseigenschaften wirkt er restriktiv für den gesamten weiteren Entwurfsprozeß.

(2) Der funktionale Entwurf ist mit Mitteln der Datenverarbeitung grundsätzlich gut unterstützbar. Er hat lediglich abstrakte Entwurfselemente zum Gegenstand, über die im Rahmen des Vorentwurfs nur einfache Berechnungen auszuführen sind.

(3) Ein Problem stellt die Integration von Werkzeugen dar, die auf der Basis der expliziten Spezifikation eine computergestützte Evaluierung von Entwurfsmodellen der anderen Entwurfsaspekte realisieren. Von besonderem Interesse ist deshalb ein Konzept zur CAAD-Systembildung aus Einzeltools.

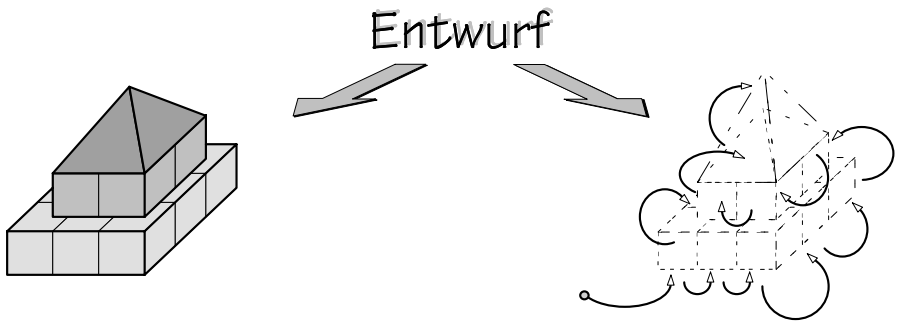
4. VORGEHENSMODELLE DES ENTWERFENS

Der Begriff 'Entwurf' wird in der Umgangssprache ambivalent genutzt. Er kann unter

- *gegenständlichem Aspekt* als Gesamtheit aller erstellten Repräsentationen verstanden werden, um ein Bauwerk fertigen zu können,

oder aber unter

- *prozessoralem Aspekt* als eine Folge von Entwurfshandlungen.



gegenständliche Sicht
Modell- bzw. Objektorientierung

prozeßorientierte Sicht
Handlungen bzw. Handlungsfolgen

Abb. 4-1 Aspekte des 'Entwurfsbegriffes'

Architekten betonen i.allg. den Handlungsaspekt. So unterstreicht auch SALZMANN die *Tätigkeit* des Entwerfens und definiert den architektonischen Entwurf in [DON88] wie folgt:

> „Entwerfen ist die zielgerichtete, wesentlich geistig schöpferische Tätigkeit des Menschen zur Veränderung der Umwelt, die auf die Vorabbildung eines architektonischen Objektes gerichtet ... ist. Die Bearbeitung erfolgt mit Hilfe von Modellbildungen als *Dokumentation* des Erkenntnisstandes und als *Arbeitsmittel* zur Gewinnung von Erkenntnissen."

Wesentlich an dieser Definition ist die Klärung der Rolle der Dokumentation, also jeder Form externalisierter Repräsentation. Ihre Erstellung ist der beobachtbare Teil der Entwurfstätigkeit und ihr materielles Ergebnis. Als solches kann nur sie Basis des gesellschaftlich notwendigen Abrechnungsprozesses von Architektenleistung sein (siehe [HOAI]). Erwünschtes *materielles* Ergebnis sind also die vergegenständlichten Informationen, die zur Genehmigung, Herstellung und Betrieb des Bauwerkes benötigt werden[1].

Entwerfen selbst ist lt. SALZMANN jedoch eine „geistig schöpferische Tätigkeit", sie entzieht sich einer direkten Beobachtbarkeit und ist auch durch Introspektive nur bedingt greifbar. Das Vorausdenken eines vorher nicht einmal als Idee existenten Artefakts macht die Kreativität dieser Tätigkeit aus. Schöpferisch und kreativ - diese Tätigkeitsbegriffe sind laut BEITZ ([BEI85]) „... je nach Betrachtungsstandpunkt entweder sehr positiv im Sinne originärer Schaffenskraft und eines Betretens von Neuland zu interpretieren oder aber etwas kritischer als wenig plan- und nachvollziehbar, als mehr künstlerisch - intuitiv und kaum erlernbar" und weiter „*Der Künstler* fragt oft nicht, ob der Betrachter, d.h. der Anwender, etwas mit seinem Werk anfangen kann. Es dient ihm vielfach nur zur Selbstbetätigung. Das darf für den Konstrukteur (und eben auch den Architekten, Anm. d.A.) nicht gelten,... ."

[1] Hierzu wird in [DRA94] GERO wie folgt zitiert:
„Das Ergebnis der Aktivität des Entwerfens ist eine Entwurfsbeschreibung... . Diese Beschreibung wird gewöhnlich graphisch, numerisch oder textuell repräsentiert. Der Zweck einer solchen Beschreibung ist es, genügend Informationen über das entworfene Artefakt zu vermitteln, um es herstellen ... zu können".

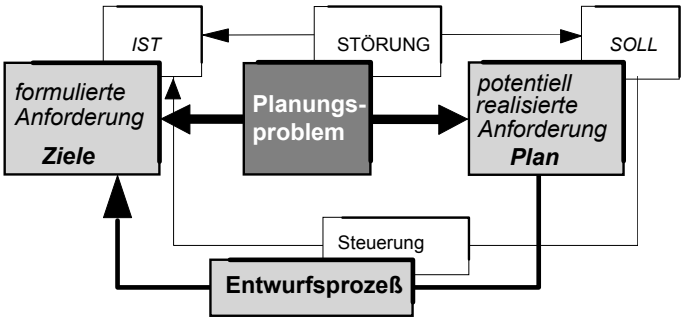
Während Modelle von Entwurfsgegenständen breiten Raum in Forschung und Technik einnehmen, sind Untersuchungen zum 'Know-How' des Entwerfens eher selten. Sie sind entweder abstrakt (siehe etwa [RIT92]), oder aber sie sind an spezielle Produktklassen oder Teilprobleme des Entwurfs gebunden und als solche wenig generisch. Ein Grund hierfür ist, daß für gegenständliche Modelle Urbilder existieren, an Hand derer die Qualität der modellhaften Abbildung bzw. die Beschreibungsmächtigkeit eines Domänenmodells geprüft werden kann. Entwurfshandlungen als menschliche Intelligenzleistung hingegen können lediglich in ihren externen Wirkungen beobachtet werden, sie hinterlassen selbst nur 'Spuren' in Form von Entwurfsobjekten. Sie sind a priori subjektiv.

Aus Sicht der Künstlichen Intelligenz gibt es zwei Zielrichtungen für die Analyse der Intelligenzleistung 'Entwurf':

- Kognitive Analyse, um die Verstandesleistung 'Entwerfen' zu verstehen sowie Herstellen der Voraussetzungen zu deren computerbasierter Simulation, um die Richtigkeit der menschbezogenen Modelle experimentell zu überprüfen
- Verfügbarmachung der Verstandesleistung 'Entwerfen' durch technische Systeme. Die technische Leistung muß dabei nicht in gleicher Weise wie die menschliche erbracht werden.

Im Rahmen des ersten Gesichtspunktes sind bis jetzt nur eine sehr allgemeine Strukturierung des Entwurfsprozesses erzielt worden sowie eine eher formale Klassifikation von Entwurfshandlungen. Von einer Simulation, also einer analogen Nachbildung des Entwurfsprozesses, ist man noch weit entfernt, sie steht nicht auf der 'Tagesordnung' aktueller Forschungsprojekte.

Unter dem Aspekt der Verfügbarmachung von Entwurfsleistung durch Maschinen wird auch heute letztlich auf ein mechanistisches Vorgehen orientiert, wie es aus der Kybernetik, bzw. dem 'Operations Research' bekannt ist (siehe etwa [STA65]) oder wie es auch im 'Means-End-Analysis-Process' Anwendung findet (in [AKI86]).



„... das Entwurfsproblem ist die Aufhebung der Diskrepanz (Störung) zwischen Istzustand und einem angestrebten Sollzustand" [JOE76]

Abb. 4-2 Entwerfen als kybernetischer Regelkreis

Dieses Vorgehen impliziert, daß bei Vorliegen der folgenden Eigenschaften

- formalisierte Darstellung von *IST*- und *SOLL*-Zustand (Ziele, Plan)
- Definierbarkeit einer Differenz zwischen *IST* und *SOLL* (Störung)
- eine Menge bekannter Aktionen zur Überführung des *IST*- in den *SOLL*-Zustand (Steuerung)

ein Automat definiert werden könnte, der die Aktionenfolge zur Lösung des Entwurfsproblemes generiert. Dies führt direkt auf den Ansatz des 'General Problem Solvers' von NEWELL, SIMON und SHAW, dessen Scheitern für 'real-world' Probleme anerkannt ist (siehe Kap.1). Die Gründe hierfür sind, daß die geforderten Eigenschaften eben nicht vorliegen (können). Weiterhin läßt es die Komplexität des Problems außer acht. Selbst wenn es sich bei dem Verfahren um einen Endlichen Automaten handeln würde (was nicht bewiesen ist), heißt das nicht, daß die Problemlösung in angemessener Zeit erzielt wird. Entwerfen gehört nicht in die Klasse von Problemen, die durch solche Universalautomaten zu lösen sind.

Wichtigstes Ergebnis der kognitiven Analyse ist, daß der Objektivierbarkeit, wie sie Grundlage eines mechanistischen Vorgehen ist, Grenzen gesetzt sind (siehe Kapitel1). Unter diesem Aspekt scheinen kybernetische Modelle des Entwurfsprozesses, wie in [JOE76], zumindest fragwürdig, da eine Analyse hier in Kategorien erfolgt, die sich an möglichen technischen Umsetzungen orientieren und die dem individuell und sozial determinierten Prozeß 'Entwurf' nicht gerecht werden.

4.1 Allgemeine Aufgabenstruktur im Entwurfsprozeß

Die Beschreibung elementarer Arbeitsvorgänge und deren Klassifikationen aus der Sicht des Architekten dient sowohl der Gliederung des Vorgangs selbst als auch der Auswahl von Werkzeugen, die diese generischen, d.h. von der jeweiligen Domäne unabhängigen Tätigkeiten unterstützen. Eine Darstellung von Tätigkeitsarten sowie eine Abschätzung ihres quantitativen Umfangs im Rahmen der durch Architekten zu leistenden Entwurfsphasen wird bei WENZEL ([WEN90]) vorgenommen. Betrachtet wurde dort das Vorgehen von Architekturstudenten im letzten Studienjahr[2].

Tätigkeit		***Zeitanteile in [%]***		
		von	Mittel	bis
A	**Informieren**	10	**19**	26
A1	Standortanalyse	3	8	16
A2	Anlaufberatungen	0	2	10
A3	Standards, Vorschriften	0	2	4
A4	bekannte analoge Lösungen	0	8	14
B	**Entwerfen**	33	**48**	59
B1	Funktionsbestimmung	8	17	29
B2	Raumanordnung, Gestaltung	5	17	35
B3	Konstruktionsprinzip festlegen	2	15	34

[2] Durch die Studenten wurden lediglich Leistungsbilder erbracht, die den Phasen 1, 2, 3 gemäß HOAI entsprechen. Die gleiche Quelle nennt für Architekten, die im industriellen Wohnungsbau mit einer Baulückenschließung befaßt waren, Tätigkeitsanteile von 5% für die Konzeption, 15% für Projektierung, 80% für Koordinierung und Überwachung.

C	**zeichnerische Dokumentation**	12	**23**	32
D	**rechnerische Prüfung, Nachweise**	0	**2**	5
E	**visuelle Prüfung, Kontrolle**	1	**4**	8
F	**Ändern**	1	**7**	18
G	**Listen erstellen**	0	**0.2**	1
H	**Schriftwechsel**	1	**2**	5

Tab. 4-1 Tätigkeitsarten und deren Verteilung in den Leistungsphasen 1-3 HOAI ([WEN90])

Die angeführten Tätigkeiten treten zwar in den Entwurfsphasen mit unterschiedlicher Häufigkeit auf, sind selbst aber nicht an spezielle Phasen gebunden. So überwiegen Tätigkeiten des 'sich informierens' in den Phasen 1 und 2 der HOAI, während die rechnerischen oder visuellen Prüfungen eher in den Phasen 3 bis 5 erbracht werden. Es fällt auf, daß diese Tätigkeitsbilder zwar für den Architekten anschaulich und intuitiv einsichtig sind, aber für eine Strukturierung von CAAD-Systemen keine Verwendung finden, soweit sie überhaupt durch CAD-Tools unterstützt sind. So ist zwar das Erstellen von Schriftwechseln sehr gut unterstützt, die Integration von Textdokumenten in ein (CAD-)Projektmodell ist jedoch kaum möglich. Module zum 'Fallretrieval' in 'Case Bases' (Tätigkeit A4) oder zum 'Vorschriftenretrieval' (A3) aus einem Entwurfskontext heraus sind bis jetzt nicht unterstützt.

4.2. Analyse des Entwurfsprozesses

4.2.1 Pragmatische Entwurfsmodelle

Pragmatische Prozeßmodelle unterteilen den idealisierten, zeitlich geordneten Entwurfsprozeß gemäß definierter Modellzustände in Entwurfsphasen. Diese Prozeßmodelle orientieren sich also eher am Zustand des Bauwerksmodells, als an den geistigen Prozessen, mit denen es erzeugt wurde. Ihr Zweck ist die zeit- und aufwandsbezogene Erfassung des Entwurfsprozesses, um ihn organisieren zu können. In diesen Phasen lassen sich Entwurfs- und Steuerhandlungen finden, die sich nach Häufigkeit und Abstraktionsniveau beträchtlich unterscheiden. Die Tätigkeiten selbst sind allerdings im pragmatischen Modell nicht Gegenstand der Betrachtung und so erlauben sie keine Aussagen über die Art der geistigen Tätigkeiten, die der Entwerfende im Planungsprozeß leistet. Die Einteilung betont einen sequentiellen Charakter des Entwurfsprozesses und gibt bei einer a posteriori Betrachtungen den Ablauf des Planungs- und Herstellungsprozeß gut wieder.
Zweck dieser pragmatischen Gliederung ist es, den Planungsprozeß so zu unterteilen, daß möglichst abgeschlossene Leistungsbilder entstehen, die sich als Basis für die Planbarkeit, Austauschbarkeit, Abrechenbarkeit von Architekten- bzw. Ingenieurleistungen eignen, wie dies ja in der 'Honorarordnung für Architekten und Ingenieure' [HOAI] vorgesehen ist.
So sind in der HOAI 9 Leistungsphasen festgelegt, von denen sich bei Gebäuden (§15) die Phasen 1 bis 7 mit der Planung und Vorbereitung des Bauwerksherstellungsprozesses befassen und die Phasen 8 und 9 mit der Überwachung und Endkontrolle des Herstellungsprozesses selbst. Neue Vorschläge erweitern diese Leistungsphasen noch um eine Phase 0,

die sich mit einer erweiterten Standortanalyse befaßt sowie die Phasen 10 und 11, die ein erweitertes Facility Management bis hin zum Abbruch zum Gegenstand haben ([KAH95]).

Insbesondere den eigentlichen Entwurfsphasen (Phasen 1 bis 5 HOAI) lassen sich Problemtypen zuordnen, die einen Leitfaden zum methodischen Entwerfen und Konstruieren geben sollen. So untergliedert etwa PHAL ([PHA77]) sein Vorgehensmodell in 'Arbeitsschritte', die zu durchlaufen sind (siehe Abb. 4-3 a, vergleiche auch [FRE85], [MOR87] oder auch [VDI85]). Diese Prozeßmodelle stammen aus den 70er Jahren und waren (und sind) hauptsächlich im Maschinenbau populär. Im Gegensatz zur Bauplanung machen die dort vorherrschenden numerisch-konstruktiven Probleme und die Verfügbarkeit klarer Aufgabenstellungen und Kausalketten für das Wirkprinzip von Konstruktionen ein solches Vorgehen sinnvoll.

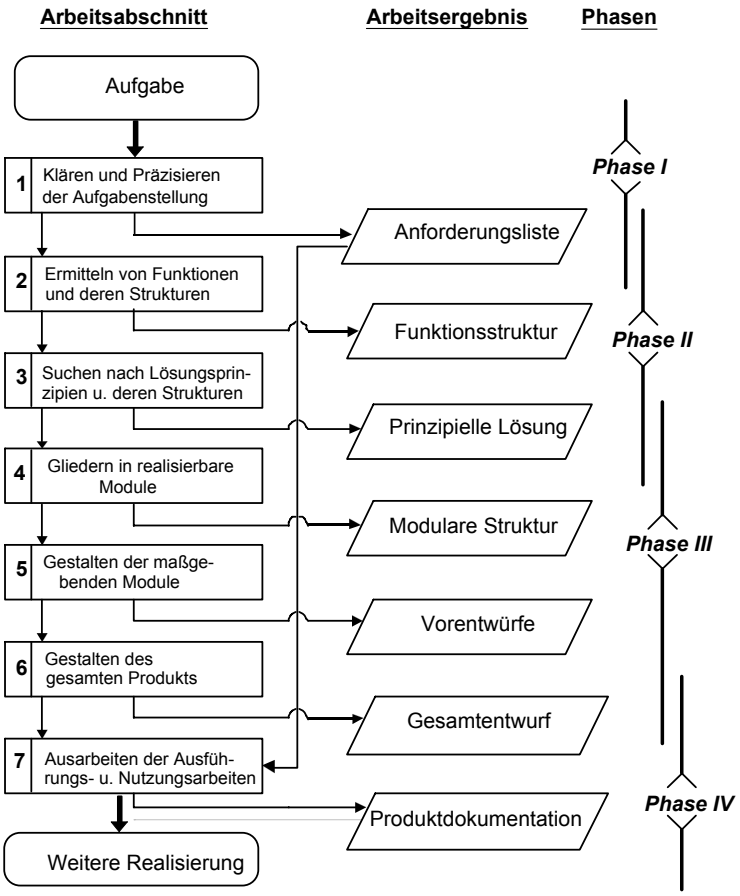


Abb. 4-3 a Generelles Vorgehen beim Entwickeln und Konstruieren, [PHA77]

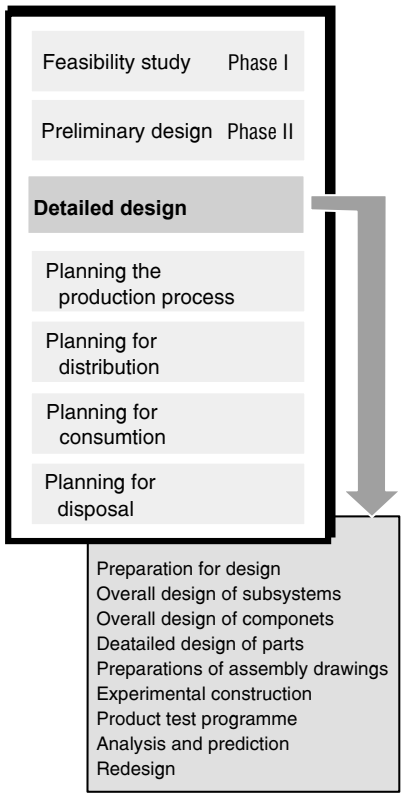


Abb. 4-3 b „Design morphology", [ASI62]

In die Gestaltung von CAD-System haben solche Betrachtungen keinen Eingang gefunden. Sie wären Grundlage für durchgängige CAD- oder CIM-Lösungen[3], die im Bauwesen aber nicht verfügbar sind. Einzelne CAD-Tools sind, soweit sie nicht generische Zeichen- oder Rechenwerkzeuge sind, bestimmten Phasen zugeordnet, wie z.B. AVA-Systeme den Leistungsphasen (5), 6, 7, (9). Bedingt durch den Produktcharakter 'Bauwerk' (siehe Kap. 1) sind solch durchgängige Lösungen jedoch nur für einzelne, stark standardisierte Produktgruppen als 'Inhouse-Lösungen' bei Komplettanbietern zu erwarten.

[3] 'Durchgängige CAD-Systeme' würden *alle* Planungsphasen unterstützen. Planungsergebnisse vorangegangener Phasen wären voll nutzbar. CIM-System würden darüber hinaus auch einen automatisierten Herstellungsprozeß unterstützen.

Phase	Leistung	Bauwerkslebensphasen	Bewertung in % der Grundleistung A	B	C
0	**Vorleistungen** Analyse, Darstellung des „strukturierten" Orts für das Bauwerk				
1	**Grundlagenermittlung** Ermitteln der Voraussetzungen der Planung		3	3	3
2	**Vorplanung** (Projekt und Planungsvorbereitung) Erarbeiten der wesentlichen Teile einer Lösung der Planungsaufgabe		7	10	7
3	**Entwurfsplanung** (System- und Integrationsplanung) Erarbeiten der endgültigen Lösung der Planungsaufgabe	Leistungsphasen nach HOAI	11	15	14
4	**Genehmigungsplanung** Erarbeiten und Einreichen der Vorlagen für die erforderlichen Genehmigungen oder Zustimmungen		6	6	2
5	**Ausführungsplanung** Erarbeiten und Darstellen der ausführungsreifen Planungslösung		25	24	30
6	**Vorbereitung der Vergabe** Ermitteln der Mengen und Aufstellen von Leistungsverzeichnissen		10	7	7
7	**Mitwirkung bei der Vergabe** Ermitteln der Kosten und Mitwirkung bei der Auftragsvergabe		4	3	3
8	**Objektüberwachung** (Bauüberwachung) Überwachen der Ausführung des Objektes		31	29	31
9	**Objektbetreuung und Dokumentation** Überwachen der Beseitigung von Mängeln und Dokumentation des Gesamtergebnisses		3	3	3
10	**Nutzung** des Bauwerks Beratung von Nutzern, Optimieren der ablaufenden Prozesse, Erfahrungsgewinn für zukünftige Projekte	Erweiterte Leistungsphasen nach [KAH95]			
11	**Stillegung** des Bauwerks Organisation von Umnutzung, Demontage, Recycling, umweltverträgliche Entsorgung				

Tab. 4-2 Leistungsphasen im Hochbau nach [HOAI] sowie KAHLEN [KAH95]
A - Gebäude, **B** - Freianlagen, **C** - raumbildende Ausbauten

Die gezeigten pragmatischen Modelle geben Leistungsphasen in einem assoziierten Abstarktionsgrad/Unschärfe des Entwurfsgegenstands gut wieder, sie lassen eine Betrachtung des Detaillierungsgrades sowie die Komplexität tatsächlich auftretender Steuerungen jedoch außer acht. Sie eignen sich zur Grobgliederung des Gesamtprozesses, eine detaillierte Untersuchung des 'Wie' des schöpferischen Entwurfs liefern sie nicht.

4.2.2 Designstrategie

Realistische Entwurfsaufgaben, selbst wenn sie gut abgegrenzt und vollständig spezifiziert vorliegen, sind im allgemeinen zu komplex, um sie in einem Zug zu lösen. In einem Zug heißt hier, daß zu *einem* Entwurfsproblem *ein* Lösungsobjekt ganzheitlich assoziiert werden kann, welches der Spezifikation genügt. Die Problemdekomposition ist eine Möglichkeit zur Beherrschung von Komplexität. Dekomposition heißt der Prozeß der Aufspaltung des Problems in einfachere, d.h. weniger komplexe Teilprobleme. Dieser Prozeß kann rekursiv so lange wiederholt werden, bis entsprechend des Abstraktionsgrades der bearbeiteten Entwurfphase eine Lösung in einem Zug bestimmt werden kann.

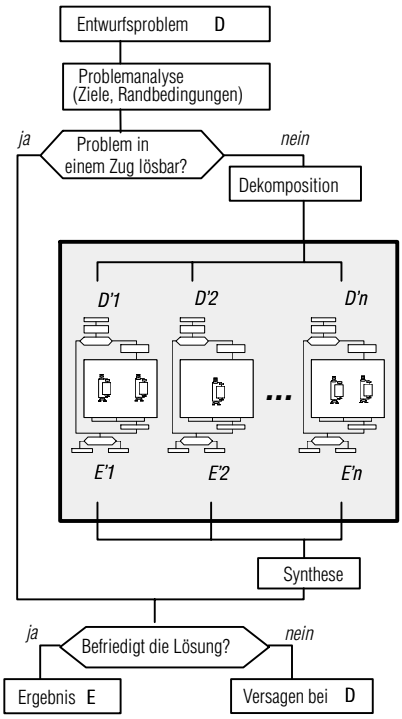


Abb. 4-4 a Rekursive Struktur des Entwurfsprozesses (theoretisch ohne Backtracking)

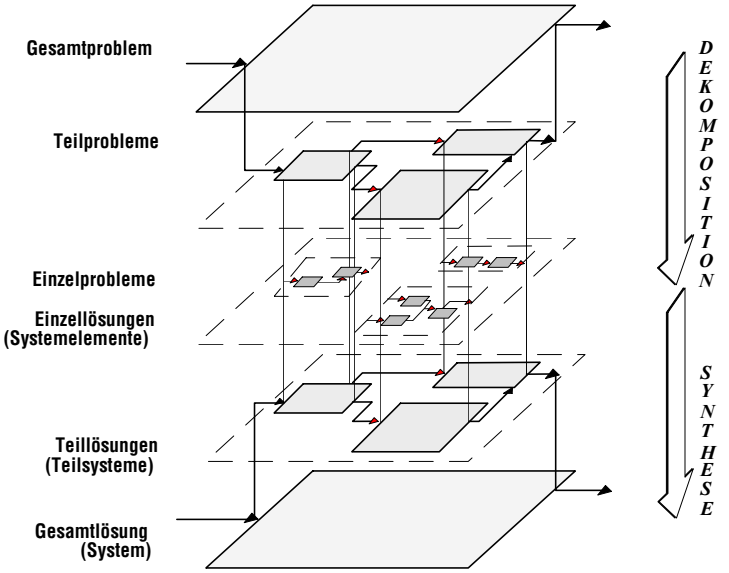


Abb. 4-4 b Problem- und Systemstrukturierung (nach [VDI85])

Diese Gliederung der Problemstruktur kann wiederum als hierarchischer Graph dargestellt werden, der im weiteren als Taskstruktur bezeichnet wird. Dieser Graph korrespondiert mit der kompositionellen Hierarchie von Modellelementen, die durch den Nutzer aufgebaut wird. Während dieser jedoch den Zustand des Bauwerksmodells zu einem bestimmten Zeitpunkt widerspiegelt, beschreibt die Taskstruktur den Entwurfsvorgang, in dem die erwartete Funktion von ansonsten noch abstrakten Objekten (Funktionselemente, Abb. 4-6) spezifiziert wird. Ein wesentlicher Teil des Problemlösungsprozesses besteht aus der Zuweisung von Realisierungselementen (Lösungselemente) zu Funktionselementen (Problemelemente).
Bestandteil jeder rekursiven Problemlösungsstrategie ist jedoch neben der Lösung des terminalen Problems die Synthese der Teillösung zur Gesamtlösung.

Als Illustration der Bestandteile eines rekursiven Problemlösungsverfahrens soll hier der simple Algorithmus zur Fakultätberechnung dienen:

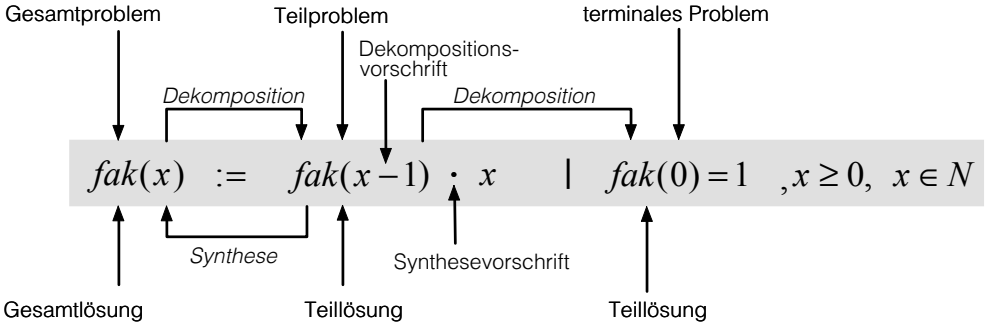


Abb. 4-5 Rekursive Problemlösungsverfahren

Wesentlicher Teil des Problemlösungsverfahrens ist also die Fähigkeit zur Synthese oder, wie in [BAK93]) formuliert, zur *Rekomposition* von Teillösungen. Für die Entwicklung von (wissensbasierten) CAD-Systemen schlägt DÖRNER ([DÖR91b]) eine Formalisierung vor, die auf die Klasse von Konfigurierungsproblemen im 'Routine Design' zugeschnitten ist und auf die hier nicht näher eingegangen wird, da die dort gültigen Prämissen nicht ohne weiteres auf den Aufgabentypus 'Baudesign' übertragbar sind[4]. Hauptproblem ist, daß diese Systeme für den Konfigurationsprozeß eine bestimmte, a priori fixierte Aufgaben- oder Taskstruktur vorsehen, da die Aufbaustruktur der zu konfigurierenden technischen Artefakte ebenfalls vorgegeben ist. Eine Funktionsbeschreibung ist meist nur implizit gegeben, sie ist Teil der CAD-Lösung selbst, d.h. das Vorgehen beim Konfigurieren folgt einer Taskstruktur, die vom Programmentwickler vorgegeben ist (vergl. [STE90]). Der Aufbau einer hierarchisch dekomponierbaren Funktionsspezifikation als Definition von Zielen für Entwurfstasks ist gegenwärtig nicht von CAD-Systemen unterstützt.

Für eine Charakterisierung des Entwurfsvorganges ist die Art und Weise, wie diese hierarchischen Strukturen aufgebaut werden, von großer Bedeutung. Am Beginn jedes Entwurfes steht eine umfassende, abstrakte Aufgabenstellung, die die Realisierung eines gesellschaftlichen Bedürfnisses zum Ziel hat. Mit dieser Entwurfsaufgabe ist der 'Top-Knoten' des hierarchischen Taskgraphen gegeben, falls diese Struktur explizit hergestellt wird. Im Laufe eines Analyseprozesses wird diese allgemeine Entwurfsaufgabe in immer präzisere Teilprobleme dekomponiert. Jedoch kann dieser prinzipiell top-down organisierte Prozeß für bestimmte Teilprobleme, die von großer Bedeutung für die Gesamtlösung sind, oder für leicht lösbare Teilprobleme durchbrochen werden. Hier werden Problemstellungen definiert und gelöst, für die ein Gesamtproblem, in das diese Teillösung strukturell einzuordnen wäre, erst noch zu definieren ist. (sieh Abb. 4-6)

Für CAD-Systeme, soweit sie überhaupt über ein wirkliches Objektmodell verfügen, wird im allg. eine Komposition von Komplexobjekten immer bottom-up organisiert – das Erzeugen eines graphischen Elementes in einem CAD-System visualisiert ein konstruktives Element, für das ein expliziter Kontext in Form einer Entwurfstask nicht existiert (siehe Abb. 4-7).

[4] dies bezieht sich auf Untersuchungen im Rahmen des FABEL-Projektes (vergl. [BAK93])

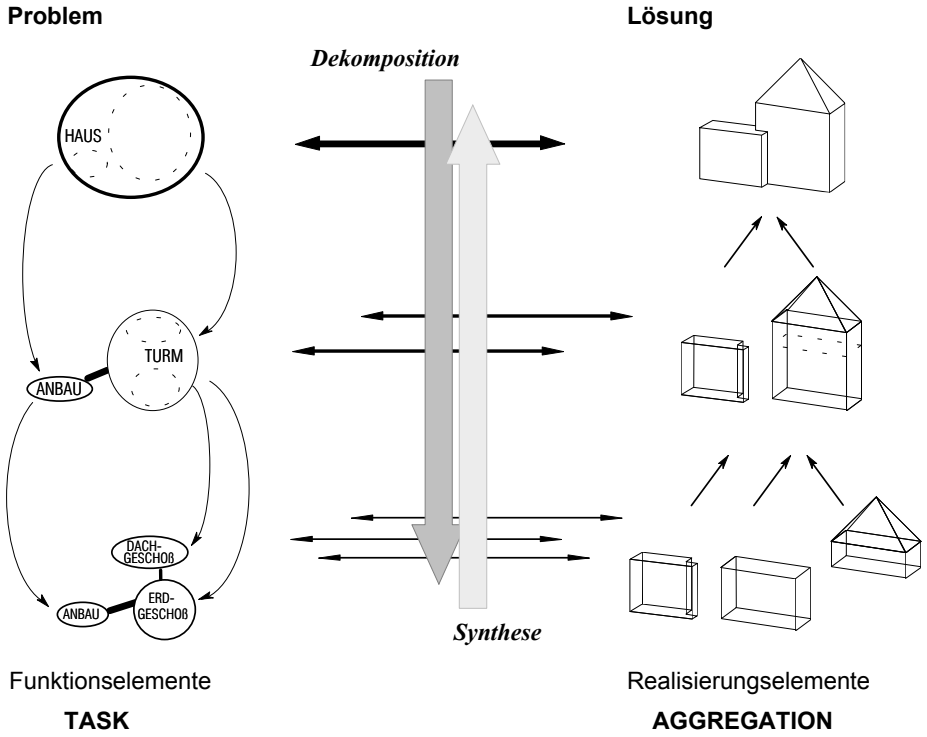


Abb. 4-6 Funktionelle Spezifikation als Dekompositionsprozeß

Diese Tasks müssen jedoch mental beim Entwerfenden sehr wohl gegeben sein, da niemand zweckfrei eine Wand in einem CAD-System erzeugt. Er weiß, diese Wand gehört zu einem Raum, der zu einer Etage gehört, die zu einem Gebäude gehört. Der Entwerfer hat also mental eine Taskstruktur aufgebaut und damit eine korrespondierende Gebäudestruktur. Genau dieser Prozeß wird aber durch CAD-Systeme nicht nachvollzogen. Dies ist auch eine Begründung, warum 'Design' nicht unterstützt wird, sondern letztlich immer 'Drafting'. Diese Technologie ermöglicht es, mehr Varianten in gleicher Zeit zu repräsentieren und anschließend die Repräsentationen zu vergleichen. Eine Hilfe bei Modellaufbau und -strukturierung erfolgt nicht.

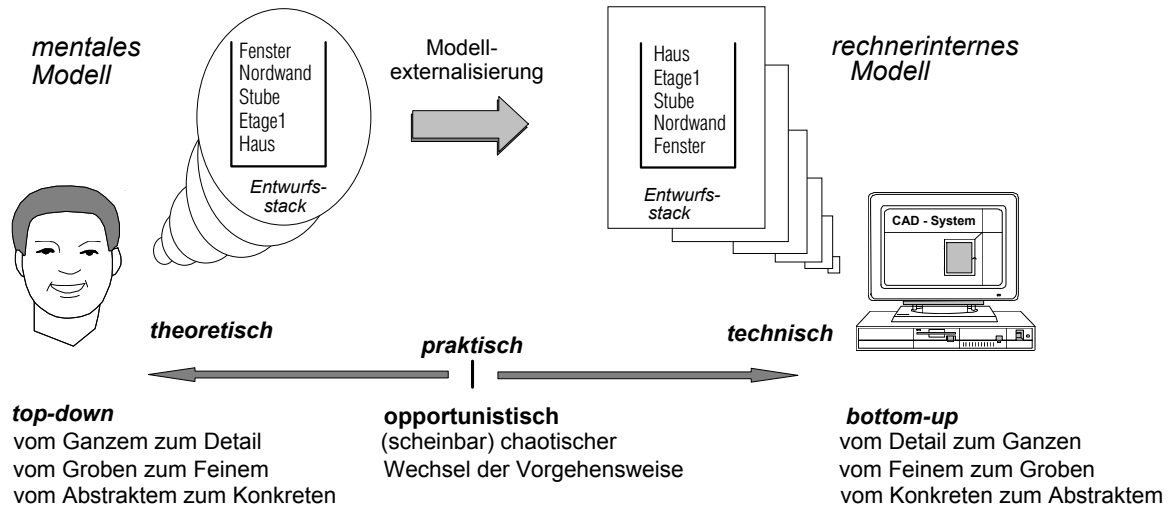


Abb. 4-7 Pole der Entwurfsstrategie

Eine analoge technische Nachbildung des top-down organisierten, mentalen Entwurfsprozesses ist jedoch ebensowenig vorstellbar. Der Entwerfende wäre gezwungen, alle (d.h. auch unbewußte) Entwurfsaufgaben in Form von Objekten zu externalisieren. Durch die hohe Subjektivität dieser Vorgänge scheint dies unmöglich, zumal schwerwiegende Aufwands- und Akzeptanzprobleme zu erwarten wären. Eine entsprechende CAD-Technologie muß deshalb eine *opportunistische* Entwurfsstrategie unterstützen, die sowohl top-down als auch bottom-up Handlungsanteile zuläßt und so dem beobachtbaren, scheinbar chaotischen Entwurfsvorgang zu folgen vermag (Abb. 4-7 und 4-8 b).

Unabdingbare Grundlage hierfür ist die explizite Existenz einer Aufbaustruktur. Das Ziel von CAD-Systemen sollte sein, Modelle im Rechner (RIM) in der gleichen Weise zu erstellen, wie das der Architekt mit seinem mentalen Modell tut. Alle beobachtbaren natürlichen Prozeßsequenzen sollten am Modell ausführbar sein. Die Verfügbarkeit des operationalen Modells, das zu einem Entwurfsergebnis führt, ist weiterhin Grundlage für leistungsfähige UNDO-Mechanismen (ungesteuertes Backtracking) oder 'Truth Maintenance Systems' (gesteuertes Backtracking [PUP91]). ENCARNACÃO fordert in diesem Zusammenhang für Systeme, die 'Conceptual Design' unterstützen ([ENC90]):

> „ ... (ist) es notwendig, daß der Designer mit einem System arbeitet, welches die Abfolge von Entwurfsoperationen in der Vielfalt der Entwurfspfade bewahrt, um irrelevante Pfade zu verwerfen oder neu aufzunehmen."

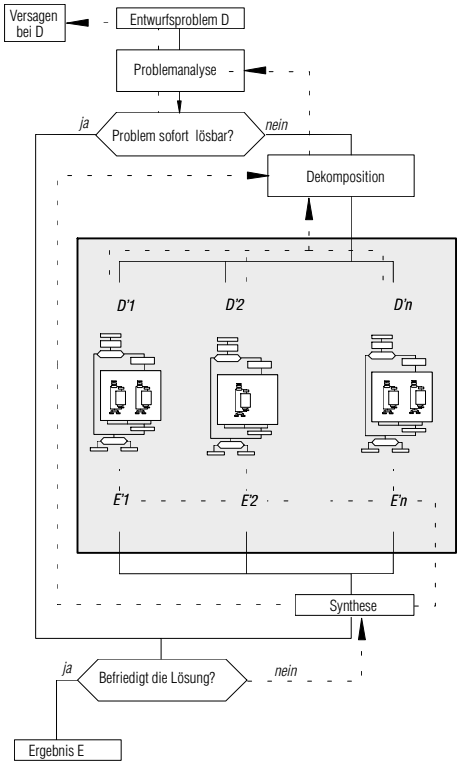


Abb. 4-8 a Rekursive Struktur des Entwurfsprozesses (mit ungesteuertem Backtracking)

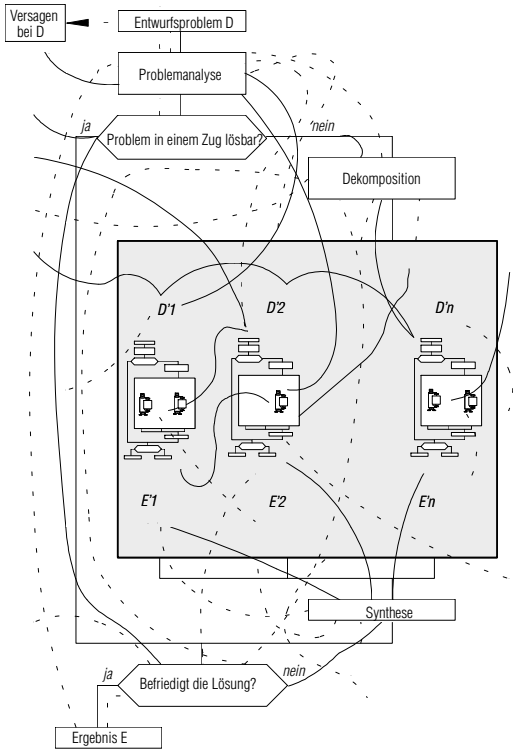


Abb. 4-8 b Realistische Struktur des Entwurfsprozesses (chaotischer Prozeßverlauf)

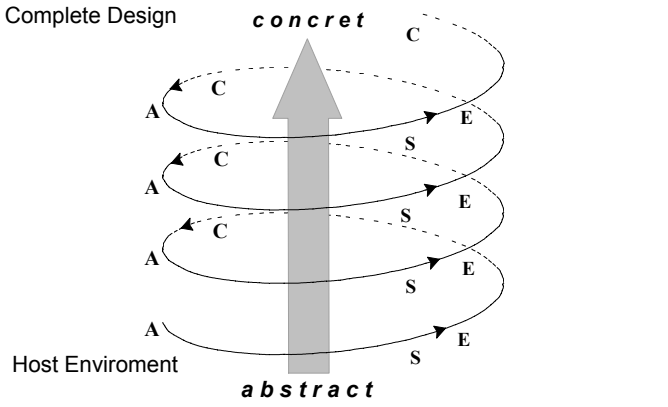
4.2.3 Kognitive Modelle

Kognitive Prozeßmodelle zielen auf ein grundlegendes Verständnis der Verstandesleistung 'Entwurf'. Eine philosophisch-psychologisierende Sicht hat deren genaue Analyse beim Menschen zum Ziel, um zu einem Modell zu gelangen, das diese Fähigkeit beim Subjekt erklärt. Begrenzte Erfolge wurden hier vor allem bei der Analyse perzeptiver Prozesse[5] innerhalb des Entwurfs erzielt. Fragen des Erkennens und Wiedererkennens von (alten) Problemstellungen sowie das Assoziieren und Adaptieren ihrer (bekannten) Lösungen sind ein Hauptelement im architektonischen, jedoch vor allem im ingenieurtechnischen Entwurf. BROADBENT ([BRO73]) spricht hier vom '*Analogical Design*'. Ihre technische Nutzung finden diese Erkenntnisse in Konzepten des 'Fallbasierten Entwerfens', wie sie u.a. für die Architekturdomäne im Projekt FABEL genutzt werden. FABEL beschäftigt sich allerdings 'nur' mit 'routine design' in einem speziellen Anwendungsgebiet.

Ein umfassendes System von Theorien liefern philosophische Betrachtungen, die sich auf abstraktem Niveau mit der Identifizierung und Klassifikation geistiger Tätigkeit befassen. Zu Beginn der 60er Jahre erfolgte – angeregt durch Forschungen auf dem Gebiet der Kybernetik und des 'Operations Research' – eine intensive Auseinandersetzung mit dem Entwurfsprozeß. Auf der 'Conference on Design Methods' (1962, u.a. in [BRO73]) nennen CHRISTOPHERSON und PAGE folgende Grundelemente des Entwurfsprozesses:

	CHRISTOPHERSON	**PAGE**
(a)	conception	analysis
(b)	realization	synthesis
(c)	communication	evaluation

Alle Autoren betonen, daß diese Elemente lediglich einer grundsätzlichen Einordnung von Tätigkeiten dienen und daß ein Entwurfsprozeß durch deren zyklische, iterative Anwendung voranschreitet. So gelingt eine vollständige Analyse (hier also eine Bauwerkspezifikation) zu Beginn eines Entwurfsprozesses beispielsweise nicht vollständig, sondern sie wird quasiparallel zur Lösung entwickelt (siehe 'innovative design'). Auf dieser Grundlage schlägt ASIMOV (Abb. 4-9, aus [ASI62]) eine Entwurfsspirale vor, wie sie heute allgemein als Charakterisierung des Entwurfsvorganges anerkannt ist (siehe etwa [MOR87]).



analysis **(A)** ⇨ synthesis **(S)** ⇨ evaluation **(E)** ⇨ communication **(C)**

Abb. 4-9 'Design Spiral' nach ASIMOV

[5] Perzeption – die bloße, den Gegenstand außer acht lassende Wahrnehmung oder Betrachtung

Nicht nur im architektonischen Entwurf, sondern auch beim Design von Software, lösen zyklische Modelle das sequentielle Wasserfallmodell ab ([YOU93]).

MARCH ([MAR84]) ordnet diese Basistätigkeiten in ein logisches System menschlicher Schlußweisen ein. Während sich Analyse und Evaluierung mit den klassischen Schlußweisen Induktion und Deduktion erklären lassen, gelingt dies für den eigentlich schöpferischen Vorgang, die Produktion (Synthese) nicht. Er erklärt ihn durch *Abduktion*, eine durch Peirce[6] eingeführte Schlußweise. Diese Inferenz schließt „auf eine mögliche Voraussetzung (Ursache) einer Tatsache (erwünschte Wirkung) unter Beachtung allgemeiner Regeln“ ([GYA94]). Im Bezug auf den Entwurfsprozeß stellen Ursachen mögliche Lösungen der Entwurfsaufgabe dar, die einer Entwurfsinterpretation ausgesetzt werden.

$$Ursache \xleftarrow[Abduktion]{} Regeln + Wirkungen$$

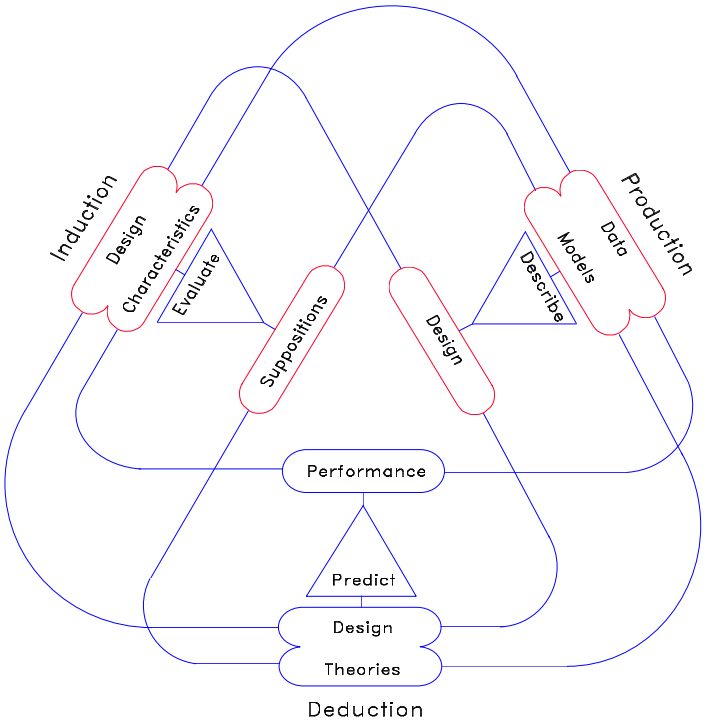


Abb. 4-10 Entwurfsmodell nach MARCH

Diese zyklische Grundstruktur des Prozeßmodells, die Basisklassifikation und kognitive Einordnung der Tätigkeiten ist heute allgemein akzeptiert. Sie wurde für den architektonischen Entwurf insbesondere durch GERO ([GER89]) verfeinert. Ein einheitlicher Sprachgebrauch zur Bezeichnung von Basistätigkeiten und deren klare Abgrenzung hat sich leider nicht herausgebildet. Zum Vergleich sei hier das Modell von TANK ([TAN91]) aufgeführt, welches das Modell von GERO aufnimmt und um die Idee der Funktionszerlegung von RICHTER ([RIC88b]) erweitert.

[6] Charles Sander PEIRCE (1839-1914), begründete die philosophische Schule des Pragmatismus

Unabhängig von der verschiedenen Bezeichnung von Basistätigkeiten des Entwurfs lassen sich die folgenden Klassen von Entwurfsaktivitäten unterscheiden:

ANALYSE Problem- d.h. Bedarfsanalyse; Herstellen einer konkreten Aufgabenstellung, von Entwurfszielen und Randbedingungen; weitgehend abstrakte Objekte

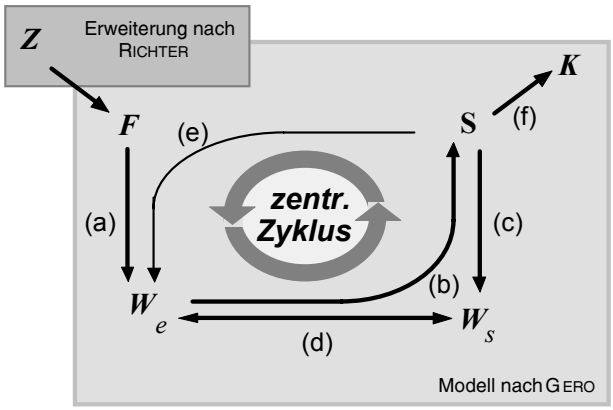
SYNTHESE Finden von Lösungen bzw. einer Varietät von Lösungskandidaten; mnemonisch besser wäre PRODUKTION, da Synthese zweckmäßiger das Zusammenfügen von Teil- zu einer Gesamtlösung bezeichnet (Aufbau der Aggregation); Ergebnis sind Objekte als modellhafte Abbilder zukünftiger Artefakte

EVALUIERUNG Erarbeitung einer Bewertung für eine Lösung, um deren Zulässigkeit festzustellen bzw. Lösungskandidaten gemäß der Zielvorstellung selektieren zu können

KOMMUNIKATION Alle Prozesse der Informationstransformation und -interpretation, um sequentielle und parallele Arbeitsteilung zu koordinieren sowie die benötigte Rückbezüglichkeit im Entwurfsprozeß zu ermöglichen.

Beschreibung des Entwurfsproblems

- A - Menge von Funktionen
- K - Designbeschreibung
- S - Struktur des Artefakts
- W_e - erwartete Wirkung
- W_s - tatsächliche Wirkung
- Z - Zweck (Bedarf)
- F - Menge von Funktionen (Spezifikation)



Basisprozesse des Enwurfszyklus

(a) Formulierung
(b) Synthese
(c) Analyse
(d) Evaluierung
(e) Reformulierung
(f) Erzeugung der Repräsentation

Abb. 4-11 zyklisches Entwurfsmodell nach GERO, RICHTER, TANK

Darüber hinaus treten Handlungstypen auf, die sich nicht mit dem gegenständlichen Bauwerksmodell, sondern mit der Organisation des Planungsprozesses befassen[7]. Auch wenn diese Handlungen keinen direkten Einfluß auf die Gestaltung des Bauwerksmodells haben, so sind sie doch mittelbar durch ihren Einfluß auf die Effizienz des Lösungsprozesses von großem Interesse für die Qualität des Bauwerksmodells.

[7] in [FOR97] wird von der 'Planung des Planungsprozesses' gesprochen

STEUERUNG Alle Aktionen, die die Abfolge von Entwurfshandlungen steuern. Sie dienen dem Ressourcenmanagement, der Organisation der Designstrategie und der damit verbundenen Variation des Abstraktionsgrades. Sie sind aus Sicht des Bauwerksmodells seiteneffektfrei.

Die Analyse kognitiver Tätigkeiten in der akademischen Forschung zielt auf eine partielle Übernahme dieser Tätigkeiten durch Software. So diskutiert etwa LIEBIG die Basisaktivitäten ([LIE93]) besonders unter dem Aspekt des architektonischen Entwurfs und untersucht die Frage, „ ... inwieweit die verschiedenen logischen Schlußweisen ihre Entsprechung in der Inferenzkomponente wissensbasierter Systeme finden (können)".
Betrachtet man jedoch die Praxis, so scheint eine direkten Adaption kognitiver Vorgehensmodelle in technischen Komponenten von CAD-Systemen gegenwärtig nicht möglich. Ihr Wert besteht eher darin, daß ein einheitliches operatives Modell '*Entwurf*' zu einer Vereinheitlichung für die Nutzung und Gestaltung von CAD-Systemen führen könnte. Die gegenständlichen Modelle, die gegenwärtig Grundlage des Austauschs von Entwurfsergebnissen sind, sollten durch operationale Anteile erweitert werden. Dies würde einerseits die Interpretierbarkeit der Modelle erhöhen. Anderseits wäre es dann realisierbar, einen suspendierten Entwurfsvorgang wieder aufzunehmen oder Entwurfsentscheidungen zurückzunehmen, die in verschiedenen Tools des Systems getroffen wurden (systemweites 'Undo').

4.3 Anforderungen an ein CAAD-Prozeßmodell

Für die Unterstützung von Entwurfshandlungen im eigentlichen Sinn müssen die Tools eines solchen Systems einem geeigneten Prozeßmodell folgen. Dieses Modell muß allgemein genug sein, um den Entwurfsprozeß für alle, besonders jedoch für die frühen Phasen abbilden zu können. Die einzelnen Tools, die diesem Prozeßmodell folgen, müssen andererseits speziell genug sein, um realistische Entwurfsmodelle der jeweiligen Domäne adäquat aufbauen zu können. Auch scheinbar chaotische Prozeßverläufe müssen mit den Tools realisierbar sein, um so die Durchgängigkeit der Systeme zu gewährleisten. Die Tools sollten eine seiteneffektfreie Rücknahme von Entwurfsaktionen gestatten. Nur dann werden sie in der explorativen Weise genutzt werden, die die Entwurfstätigkeit des Architekten kennzeichnet. Um dies zu gewährleisten, müßten Entwurfshandlungen als operativer Teil des gegenständlichen Modells gespeichert werden. Dieses Vorgehen bildet eine Voraussetzung für die Implementation leistungsfähiger UNDO- sowie ATMS- bzw. JTMS-Mechanismen[8].

Der Ausgangspunkt für die Gestaltung des operativen Modells bleibt das gegenständliche Bauwerksmodell und das ihm zugrundeliegende Modellstrukturierungsparadigma. CAAD-Tools sollten Aktionen bieten, die sich am Modellerstellungs- und -veränderungsprozeß orientieren. Diese Aktivitäten können in einem System von Designaktionen klassifiziert werden, die generell oder doch zumindest für eine bestimmtes Modellstrukturierungsparadigma generisch sind. Der Vorteil wäre, daß alle Tools, soweit sie sich derselben Strukturierung des gegenständlichen Modells bedienen, auf einem gleichen oder doch ähnlichen Satz von Modellierungsfunktionen arbeiten könnten. Bei Beibehaltung des Metamodells sind zumindest

[8] TMS - 'Truth Maintanance System', sind intelligente Techniken zur konsistenzerhaltenden Rücknahme von Schlußfolgerungen bzw. Entscheidungen. Beim JTMS (Justification based TMS) werden Begründungen einer Entscheidung gespeichert. Beim ATMS (Assumtion based TMS) können Basis-*annahmen* für Entscheidungen verwahrt werden [PUP91].

diese modellverändernden Basisaktionen generisch in allen Phasen und könnten so zu einem einheitlichen Bedienkonzept führen. Dies verfolgt letztlich das gleiche Anliegen wie Style Guides zur Erstellung einheitlicher GUI's[9] (siehe etwa Apple Style Guide [APP90]).

Zusammenfassend seien hier noch einmal Anforderungen an ein Prozeßmodell formuliert. Mit Ausnahme der weitergreifenden Aussage in Stichpunkt 2 besitzen die Feststellungen Relevanz auch für aktuelle CAD-Implementationen bzw. Modellverwaltungstools.

(1) Tätigkeitsarten und Entwurfsstrategien wechseln sich im Verlauf des Entwurfsprozesses nichtvorhersagbar ab. Dies bedingt, daß entsprechende CAD-Unterstützungen in Komponeten gegliedert sein sollten, die jeweils einzeln benutzbar sind, um solch scheinbar chaotischen Prozeßverläufen folgen zu können.

(2) Einzeltools eines CAD-Systems sollten ihre spezifische Funktionalität aus kognitiv begründeten Basistätigkeiten ableiten bzw. zusammensetzen. Durch die Annahme eines generischen Prozeßmodells wird eine Homogenisierung der Funktionalität von CAD-Tools möglich.

(3) Eine echte Entwurfsunterstützung muß neben dem technisch orientierten 'bottom-up'-Entwurf auch den mental vorherrschenden 'top-down'-Entwurfs ermöglichen. Als Implementationsgrundlage hierfür müssen entsprechende Modellverwaltungssysteme eine echte Aggregationsstruktur bieten ('part_of' / 'has_part' Relation).

(4) Ein generisches Prozeßmodell bildet die Grundlage für funktionale Erweiterungen von Objektmodellen. Die Verfügbarkeit von Entwurfsaktivitäten im Objektmodell ermöglicht die Implementation von intelligenten Rücknahmemechanismen über Toolgrenzen hinweg.

(5) Aus pragmatischen Gründen müssen CAD-Systeme an Phasengrenzen des Prozeßmodells (HOAI-Phasen) standardisierte Repräsentationen erstellen können.

[9] Graphical User Interface

5. VORSCHLAG EINES GENERISCHEN CAAD-PROZEßMODELLS

Als Basis für CA(A)D-Tools werden im weiteren generische *atomare Designaktionen* vorgeschlagen. Ziel dieses Vorschlags ist es, die Funktionalität der phasenspezifischen Tools einheitlich zu strukturieren. Die Entwurfsaktionen werden den Handlungstypen zugeordnet, die im Rahmen der kognitiven Analyse diskutiert wurden. Für eine weitergehende Klassifikation der Entwurfsaktivitäten ist ihre Stellung zum gegenständlichen Entwurfsmodell von Bedeutung. Die eigentlichen schöpferischen Designaktionen sind solche, die zur Konstituierung des deskriptiven Bauwerksmodells führen. Während sich also modellverändernde Aktionen mit dem primären Objektmodell (Ω) befassen, schaffen modellauswertende Aktionen unter Nutzung von Kontextmodell (C) und Performance-Modell (P) eine Bewertung x. Diese Bewertung qualifiziert die Gültigkeit oder quantifiziert die Qualität des Objektmodells (siehe Kapitel 2). Das primäre Objektmodell bleibt hierbei unverändert. Die Entwurfsaufgabe selbst wird durch die Verfügbarkeit einer Bewertung sehr wohl verändert. Aktionen, die der Organisation des Lösungsprozesses selbst dienen, sind keine Entwurfsoperationen im eigentlichen Sinne. Sie haben jedoch mittelbar Einfluß auf den Zustand des Entwurfsproblems, da sie die Effizienz des Lösungsprozesses wesentlich bestimmen. Somit ergibt sich die in Abbildung 5-1 dargestellte Klassifikation von Entwurfsaktionen, in die bereits die im weiteren behandelten 'atomaren Designaktionen' eingeordnet sind.

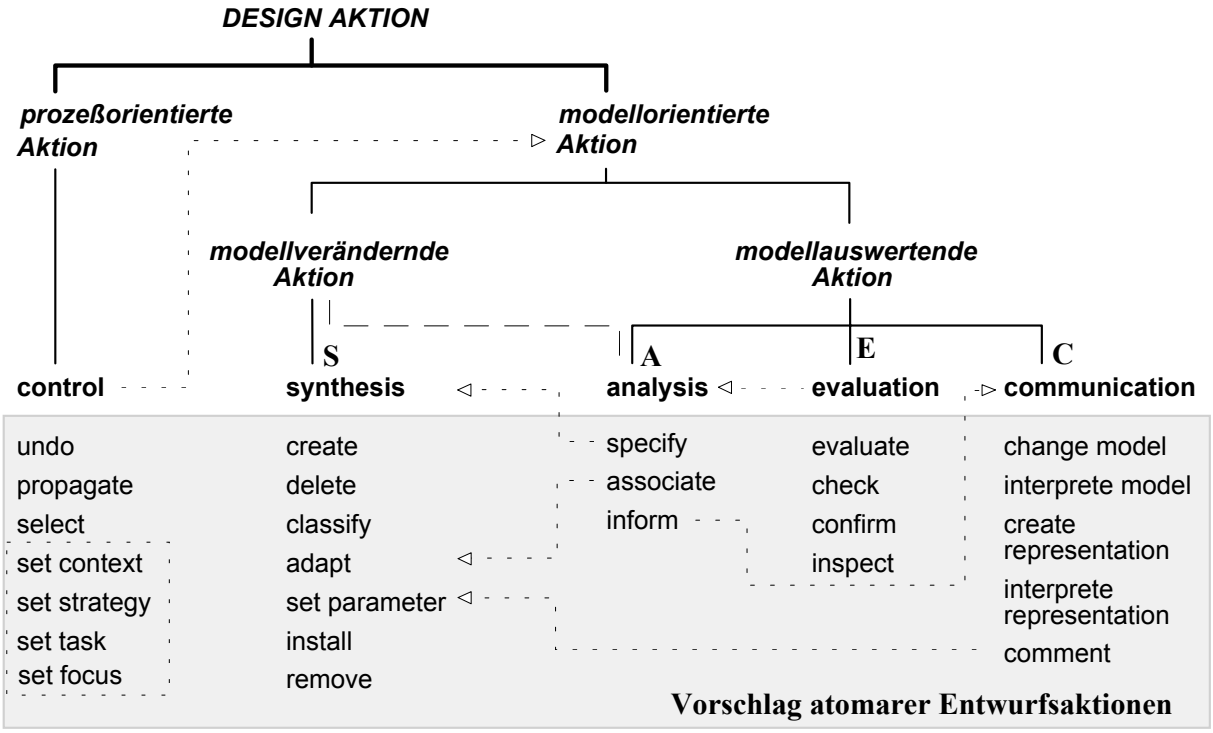


Abb. 5-1 Atomare Entwurfsaktionen und ihre Klassifikation

Je nach Art des einzelnen CAD-Tools (z.B. 'design automat') und des geplanten Einsatzgebiets werden die Tools eine bestimmte Teilmenge der kognitive begründeten Funktionalität realisieren. Für die Bildung von CAD-Systemen bleibt die vertikale, pragmatische Gliederung des Planungsprozesses in Phasen die bestimmende Strukturierung. Dies resultiert aus der erforderlichen Austauschbarkeit und Abrechenbarkeit von Entwurfsergebnissen, die Folge der Spezialisierung und Arbeitsteilung sind. So sind verschiedene Domänenmodelle für verschiedene Entwurfsphasen mit einem jeweils

spezifischen Grad von Abstraktion bzw. Unschärfe Ausdruck dieser Spezialisierung. An den Phasengrenzen, wie sie etwa die HOAI vorsieht, muß es möglich sein, jeweils konsensfähige (im Idealfall standardisierte) Repräsentationen zum Modellaustausch zu erzeugen oder aber zu interpretieren. An diesen Grenzen findet Kommunikation statt.

Jedes Tools selbst sollte aber die horizontale kognitive Gliederung des Planungsprozesses nutzen. So entsteht ein einheitliches, spiralförmiges Prozeßmodell des Entwurfs (Abb. 5-2).

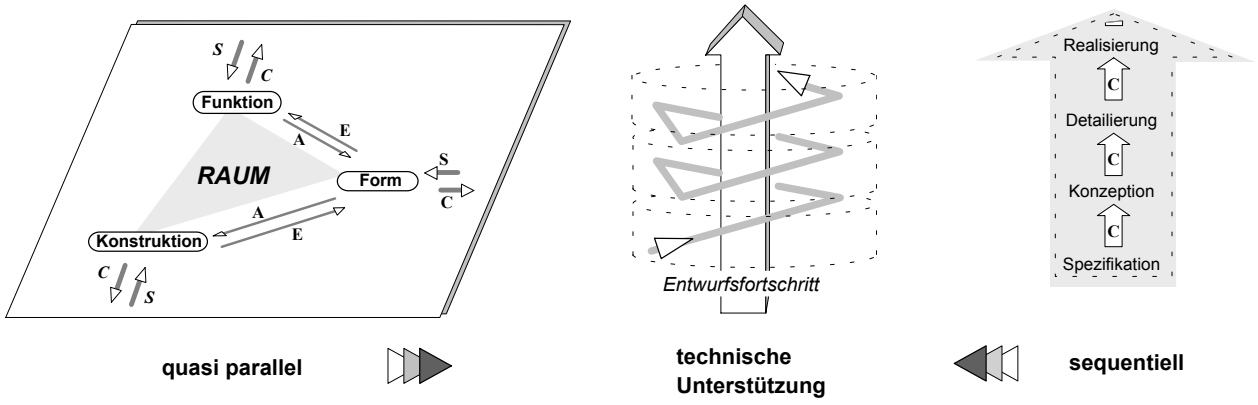


Abb. 5-2 Vereinheitlichtes Prozeßmodell des Entwurfs

Die im weiteren vorzustellenden Designaktionen erheben keinen Anspruch auf Vollständigkeit bzw. vollständige Orthogonalität, da derartige Untersuchungen den Rahmen der Arbeit gesprengt hätten. Sie dienen der Illustration einer möglichen Toolstrukturierung im Hinblick auf das vereinheitlichte Prozeßmodell. Einen Schwerpunkt bilden hierbei synthetisierende Tätigkeiten, da sie für CAD-Tools der Kategorie ‘design repository’ von besonderem Interesse sind. Es werden bei den Entwurfsaktionen jeweils Beispiele angeführt.

5.1 Synthese

Synthetisierende Aktionen (Produktionen) sind Aktionen, deren Anwendung das Bauwerksmodell erzeugen oder verändern. Sie sind somit Werkzeuge eines schöpferischen Akts. Mittels dieser Operationen werden Lösungen bzw. eine Varietät von Lösungskandidaten erzeugt. RITTEL bezeichnet Varietätserzeugung (Synthese) und Varietätseinschränkung (Evaluierung ↓) als Elementartätigkeiten des Entwurfs ([RIT92]). Jedes CAD-Tool, das ‘echte’ Modelle aufbaut, verfügt notwendigerweise explizit oder implizit über Syntheseoperationen – explizit, wenn mit ihnen durch den Nutzer Modelle kreiert werden, implizit, wenn das Tool durch Interpretation eines externen Modells ein eigenes, internes Modell aufbaut. Syntheseoperationen sind im Sinne der Programmierung seiteneffektbehaftet. Ihre Ausführung bewirkt eine Änderung des Modellzustandes.

Produktionen sind zielgetrieben. Sie werden durch den Entwerfenden zweckgebunden zur Lösung einer Entwurfsaufgabe eingesetzt. Für eine CAD-Entwicklung gelingt jedoch eine formale, vollständige oder auch nur hinreichende Abbildung von Zielen (Funktionselementen) zu Lösungen (Realisierungselemente) nur für sehr spezielle Entwurfsaufgaben (siehe [STE90]).

Als abduktive Prozesse entziehen sie sich einer allgemeinen Algorithmierbarkeit, ihr Einsatz ist somit weitgehend nutzerbestimmt.

$$Funktionselemente \xrightarrow{Abduktion} Realisierungselemente$$

In der Untersuchung von WENZEL (vergl. Tabbelle 4-1) handelt es sich um die Tätigkeiten B2 – 'Raumanordnung, Gestaltung', B3 – 'Konstruktionsprinzip', F – 'Ändern' mit einem Gesamtzeitanteil von 39%.

Die im folgenden vorgeschlagenen modellverändernden Syntheseaktionen gehen von der Annahme aus, daß ein objektorientiertes Modellierungsparadigma Verwendung findet.

***Create**:* Erzeugen einer Instanz allgemeinsten Typs, d.h. eine Instanz wird immer als Ausprägung einer umfassenden Basisklasse erzeugt. Diese Basisklasse kann gemäß eines Kontextes variiert werden.
- Erzeuge ein 'Objekt213'
- Erzeuge ein 'Bad'

***Delete**:* Löschen einer Instanz oder Streichen eines alternativen Lösungskandidaten (i.allg. beim Backtracking oder anderen Kontrollaktionen). Bei Objekten, die über eine Aufbaustruktur verfügen, wird eine Strategie zur Behandlung der Teile benötigt.
- Lösche den 'Anbau',
- „Auch 'Schieferdeckung' kommt nicht in Frage"

***Classify**:* Präzisieren oder Abstrahieren durch Typwechsel in einem Zweig der Taxonomie, bzw. Umklassifizieren einer Instanz
- 'Bad1' ist ein 'Hygienebereich', Später wird 'Bad1' ein 'Wannenbad'
- Es erfolgt eine Umnutzung von 'Bad1' als Küche

***Adapt**:* Komplexe Operation des Erzeugens von Realisierungselementen, die mit bekannten Funktionselementen assoziiert sind und deren Anpassen an die vorliegende Problemsituation.
- Generiere Objekte für Problem 'Toilettenanlage' als Kopie der Lösung aus Projekt 'Residenzhotel' und passe auf Auslegung für 120 Gäste an.

***Install**:* Untersetzen einer Instanz durch (weitere) Teile.
- 'Bad' hat 'Waschbecken' als Teil

***Remove**:* Entfernen von Teilen aus einem Ganzen
(Achtung: es werden *keine* Objekte gelöscht (Delete))
- 'Wickelraum' sei nicht mehr Teil des 'Sanitärtrakts'.

Install und *Remove* sind die zwei Basisfunktionen zum Erstellen der Aufbaustruktur. Sie erstellen lediglich Kanten dieses heterarchischen Graphen, nicht dessen Knoten (erfolgt durch *Create*). Sie sind von großer Bedeutung für die Realisierung der Aktionen *Refine* und *Aggregate* (Abb. 5-3) im Sinne von Entwurfsstrategien. Diese beiden Aktionen lassen sich aber durch eine Folge von *create-install* Aktionen darstellen, d.h. sie gehören nicht zum Satz atomarer Syntheseoperationen.

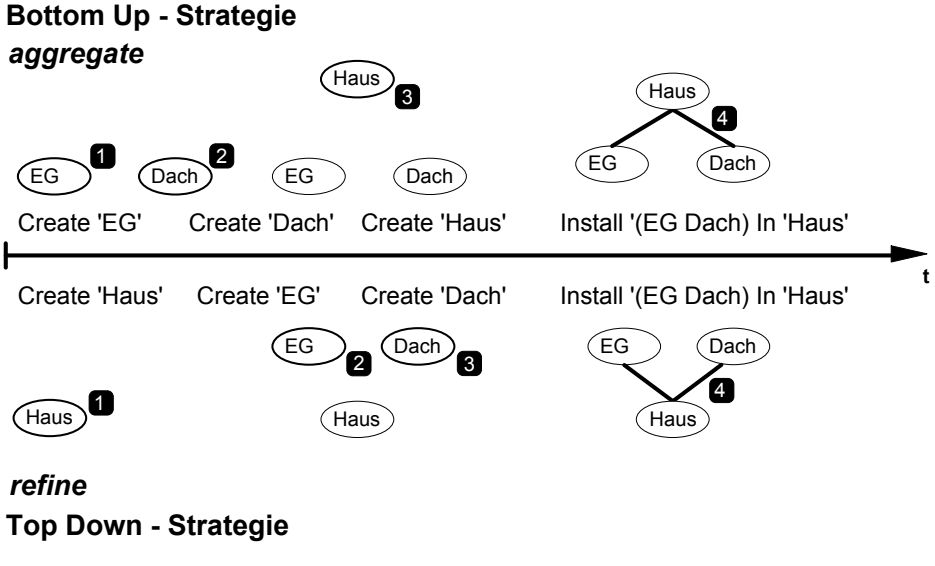


Abb. 5-3 Entwurfsstrategie als Abfolge von Syntheseoperationen

***Set Parameter**:* Verändern atomarer Modelleigenschaften, wie Setzen von Attributwerten sowie das Herstellen von Relationen und das Umbenennen von Objekten. Diese Aktionen müssen ggf. Unschärfe tolerieren.

- 'Bad' hat die 'Lage' {'Ost oder 'Nord} (unscharfe Selektion)
- Die 'Fläche' von 'Bad' hat die 'Einheit' '$m^{2,}$
  (Setzen einer Annotation (Facette) eines Attributes)
- 'Bad' ist 'verbunden' mit {'Flur'} (Herstellen einer Relation)

5.2 Analyse

Voraussetzung für den abduktiven Prozeß der Produktion von Lösungskandidaten ist nach GYALOKAY die Formulierung von „... erwünschten Erscheinungen" als den Randbedingungen (constraints) und Ziel- bzw. Optimierungskriterien einer Entwurfsaufgabe. Der Analyseprozeß ist also ein vorbereitender Prozeß des eigentlichen Entwurfs, der eine Problem- oder Bedarfsanalyse durchführt und eine konkrete Aufgabenstellung erzeugt.
In der Untersuchung von WENZEL (siehe Tab. 4-1) handelt es sich um die Tätigkeiten A1 – 'Standortanalyse', A2 - 'Anlaufberatungen', A3 – 'Standards, Vorschriften', A4 – 'bekannte analoge Lösungen' und B1 - 'Funktionsbestimmung' als der eigentlichen Fixierung der gefundenen Anforderungen. Sie umfassen einen Zeitanteil von 37%. Im Rahmen des frühen architektonischen Entwerfens kommt diesen Designaktionen also eine große Bedeutung zu, da sie keineswegs trivial sind und einen erheblichen Anteil an Arbeitszeit binden. RITTEL formuliert: „Das Problem beim Planen ist, zu verstehen was das Problem ist" ([RIT92]). Für eine systematische Entwurfstätigkeit (als Basis eine CAD-Stützung) und für die Organisationen einer kollektiven Entwurfstätigkeit ist eine Externalisierung dieser Aufgabenbeschreibung notwendig oder doch zumindest sinnvoll. Diese Externalisierung erfolgt wiederum in einem Modell, zu dessen Aufbau synthetisierende Aktivitäten erforderlich sind. Da mit Analysetätigkeiten das 'Kontext'- und das 'Performance-Modell erzeugt werden, sind sie im Hinblick auf das eigentliche Modell des zu planenden Artefakts *nicht* modellverändernd. Eine vollständige Spezifikation gelingt nicht, vielmehr erfolgt auch die Problemanalyse in einem rekursiven Prozeß, der (meist) Top Down bzw. (seltener) Bottom Up organisiert ist. Für

solch unvollständige Spezifikationen werden i.allg. bereits Lösungskandidaten generiert, d.h. Analyse und Synthese erfolgen quasiparallel.
Die Analyse bedient sich einer induktiven Schlußweise. Sie leitet mit den Randbedingungen die für den jeweiligen Entwurf gültigen Regeln ab. Hierzu wird eine Vielzahl bekannter Paare von Entwurfsproblem und Entwurfslösung verwandt. COYNE erklärt Analyse als Lernprozeß, bei dem Wissen W_i zum Erzeugen einer Designinterpretation I (Bewertung) aus Fällen $\{D\}$ (alte Designs) mit bekannter Interpretation durch Induktion abgeleitet wird. Diese Erklärung zielt auf die Erzeugung des 'Performance'-Modells.

$$W_i = f_{Induktion}(\{D\}, I) \qquad \text{([COY90])}.$$

Da der Vorrat an bekannten Entwurfsproblemen vom Subjekt abhängig ist, sind auch induktive Schlüsse notwendigerweise subjektiv. So läßt sich bei induktiven mathematischen Beweisen formal zwar die Richtigkeit des prozeduralen Vorgehens nachweisen, die Anwendung des Verfahrens garantiert jedoch keinen Erfolg. In Bezug auf die der Induktion zugrundeliegenden (Problem-) Fälle, sind die Analyseaktionen modellauswertende Aktivitäten.

Specify: Es erfolgt das Definieren eines Anforderungsvektors für aggregierte oder atomare Modelleigenschaften der aktuellen Abstraktionsstufe. Dieser Anforderungsvektor ist i.allg. (aber nicht immer) mit einem Objekt des primären Modells assoziiert. Anforderungen können sowohl Zielfunktionen als auch Constraints für Entwurfsparameter sein. Anforderungen, die die modellierte bzw. formalisierte Domäne verlassen, werden als informale Anforderung verwaltet.

- verfügbares Kapital für 'Haus' ist 450 TDM´ (Restriktion einer Zielgröße)
- „Der Bebauungsplan ist einzuhalten" (informale Restriktion)
- minimale Bauzeit für 'Haus' (Angabe eines Zielkriterium)
- Gebäudehöhe für 'Anbau' maximal 15 m (Restriktion eine Modellparameters)
- Grundfläche je 'Etagen zwischen 65 m^2 und 80 m^2
  (Unscharfe Restriktion eine Modellparameters)
- 'Etage1' und 'Etage2' sollen zusammen etwa 150 m^2 Grundfläche haben
  (Bem.: hierfür kein einheitliches Objekt im Primärmodell angebbar)
- Beleuchtungsbewertung soll anhand der Raytracingergebnisse erfolgen
  (Restriktion durch Verfahrensvorgabe).

Associate*:* Diese Aktivität erzeugt eine Problemstellung als Analogieschluß. Sie setzt eine Adaption analoger Problemstellungen im neuen Kontext voraus (Syntheseaktion). Es handelt sich um einen zielgerichteten Prozeß, bei dem der Suchraum für Analogien bereits eingeschränkt ist. Dies läßt sich technisch mittels 'Fallbasiertem Schließen' realisieren, wenn die Probleme stark formalisiert sind, was in frühen Phasen i.allg. jedoch nicht der Fall ist.[1] Assoziierende Aktionen suchen frühere Entwurfsobjekte (Lösungen), deren Ausprägung als grobe Spezifikation noch unbekannter Entwurfsobjekte in der aktuellen Entwurfsaufgabe dienen können (Bsp.: „gestalte ein Modellelement so, wie in Projekt X"). Eine andere Variante wäre die Suche von ähnlichen Grobspezifikationen in früheren Analysemodellen zur Übernahme und Anpassung der dortigen Subspezifikationen (siehe Abb. 5-4).

[1] Eine allg. Übersicht zum 'Case-Based Reasoning' gibt KOLODNER [KOL93].

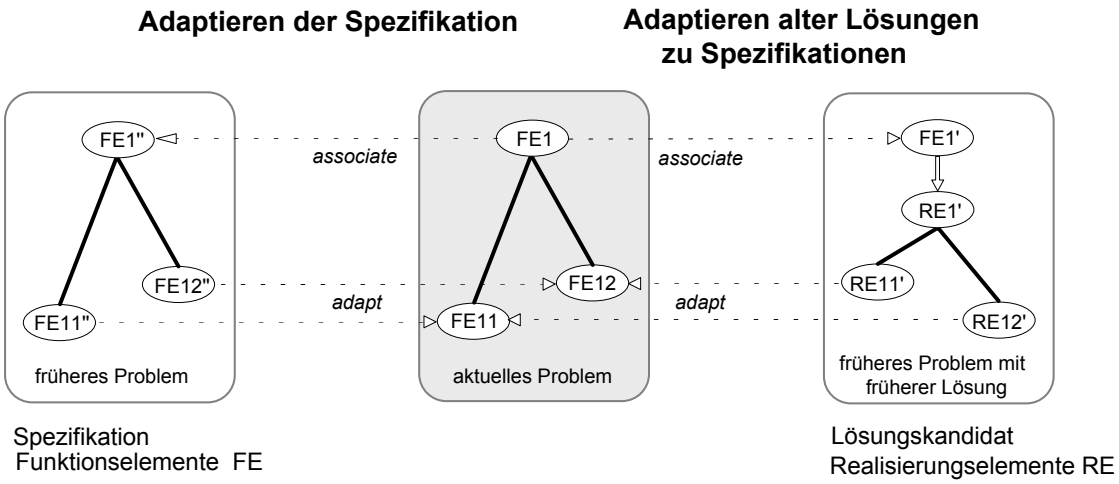


Abb. 5-4 'Assoziiere-Adaptiere'-Prozeß in der Analyse

- Gestalte 'Bad' so wie im Projekt 'Mayer'
- Gib mir alle ähnlichen 'Bäder' zur aktuellen Spezifikation des 'Hygienebereichs'.

***Inform**:* Im Gegensatz zu *Associate* durchmustert *Inform* einen Problemkontext aus formalem *und* informalem Wissen sowie bekannten Lösungen explorativ und sucht nach einer Problemstrukturierung. Es handelt sich daher um ein mehr oder weniger zielgerichtetes Sammeln von Informationen[2]. Diese Informationen werden mit *Specify* in einen Anforderungsvektor übersetzt ('Kontextmodell'). Darüber hinaus ermittelt *Inform* auch grundsätzliche technische Machbarkeiten und spezifiziert Verfahren zur Bestimmung von Zielgrößen ('Performancemodell')

- gib mir DIN Vorschriften zu 'Treppen'
- suche nach Brandschutznachweisen im Zusammenhang mit Denkmalschutz.
- Welche (Bau-)Verfahren sind für 'Wendeltreppen' anwendbar?
- Welche Ressourcen sind verfügbar zum Bau dieser 'Garage' ?
- Welche Ressourcen waren schon mal beim Bau von 'Garagen' von Bedeutung?
- Welche technischen Gütemerkmale beschreiben die Beleuchtungsqualität?
- Welche Verfahren zur Beleuchtungsbewertung gibt es ?

5.3 Evaluierung

Evaluierungen finden statt, um zu prüfen, ob Produktionen Lösungskandidaten generiert haben, die der Spezifikationen genügen. Im Rahmen einer formal logischen Betrachtung des Entwurfsprozesses können die spezifizierten Constraints als gültige Regeln eines Entwurfsproblemes betrachtet werden, während, wie beschrieben, die Lösungskandidaten als Hypothese interpretiert werden können. Evaluierung ist damit ein deduktiver Prozeß, der die Gültigkeit der Hypothese im Rahmen der gültigen Regeln feststellt ([COY90]).

$$Realisierungselemente \xleftarrow[Deduktion]{} Funktionsselemente$$

[2] vergl. KUHLEN ([KUH91]), er klassifiziert Suchformen in (informalen) Hypertexten nach ihrer 'Gerichtetheit'. *Inform* ist vergleichbar mit dem 'ungerichteten' oder 'assoziativen Browsing'

Während die 'klassische Logik' lediglich eine Verifikation der Hypothese vorsieht, lassen funktionale Erweiterungen (wie dem LAMBDA-Kalkül nach CHURCH[3]) auch eine Validierung zu und gestatten somit in einem Optimierungsprozeß das Ordnen einer Menge von Lösungskandidaten. Die Funktionselemente als Ergebnis der Spezifikation integrieren Bewertungswissen, da sie auf der Interpretation basieren, mit der sie aus einer Fallbasis abgeleitet wurden.

$$Güte := f_{Deduktion}(Funktionsselemente, Realisierungselemente)$$

Auf Grund der gut formalisierten Konzepte, die einer Evaluierung zugrunde liegen (wie Logik und Optimierung), lassen sich Evaluierungsprozesse besser technisch unterstützen als Analyse oder Synthese. Eine generelle Unterstützbarkeit ist jedoch keineswegs gegeben. Als Probleme seien hier beispielhaft genannt, daß

- Funktions- und Realisierungselemente formalisiert sein müssen, um die Evaluierung durch Software zu ermöglichen.
- Funktionselemente und Realisierungselemente einer gemeinsamen Modellstrukturierung folgen müssen, um eine Zuordenbarkeit zu gewährleisten.
- Deduktion im logischen Sinne Monotonie voraussetzt, die bei der Evaluierung im Entwurfsprozeß nicht gegeben ist.
- Gütekriterien im allgmeinen nicht skalar sind, so daß eine eindeutige Vergleichbarkeit von Lösungskandidaten nicht gegeben ist.

Bei WENZEL (Tab. 4-1) sind zu dieser Klasse von Designaktivitäten die Tätigkeiten D – 'rechnerische Prüfung, Nachweise' und E – 'visuelle Prüfung, Kontrolle' mit einem Gesamtzeitanteil von nur 6% zu zählen. Ihr wahrer Anteil dürfte aber höher liegen, da sie implizit in den eigentlichen Entwurfstätigkeiten (B2, B3) enthalten sind. In späteren Planungsphasen dürfte darüber hinaus ihr Anteil erheblich größer sein.

Evaluate: Bestimmen (Validierung) einfacher oder aggregierter Bewertungskennzahlen, zielgetriebene Bewertung formaler (Ziel-) Größen.

– Berechne den Preis, Bewerte den Energieverbrauch

Check: Verifikation in Form von Plausibilitätskontrolle, Konsistenzkontrolle, Kontrolle spezifizierter atomarer Modelleigenschaften, wie Attributwerte und Gültigkeit von Relationen.

– Entspricht die Ausprägung von 'Bad2' den Randbedingungen seiner Klasse ?
– Entspricht 'Bad2' den Randbedingungen seiner funktionalen Spezifikation?

Confirm: Informale Bewertung von Modellen, etwa bei gegebenen informalen Spezifikationen, Bewertung manuell, Pendant zu *Inform.*

– Bestätige, daß Brandschutz für 'Anbau' eingehalten ist.

Inspect: informale, vage Bewertung von Modellen ohne explizite Spezifikation, Bewertung manuell, ohne verfügbaren Algorithmus, Anwendung von Simulations- und VR-Techniken, Pendant zu *Inform.*

– (Subjektive) Bewertung des 'Raumgefühls' durch Kamerafahrt im gerenderten Modell.

[3] A.CHURCH (geb. 1903) erweiterte die Prädikatenlogik um ein Funktionenkonzept. Eine universale λ-Funktion erhält als ein Argument einen Funktionsterm sowie weitere Variable. Im Fall der klassischen Logik ergibt die Evaluierung von Ausdrücken (Goals) wahr oder falsch, im Funktionenkalkül ergibt eine ebenso universelle Auswerteprozedure ein beliebiges Rückgabeobjekt. (vergl. [STO80])

5.4 Kommunikation

Kommunikation ist keine Entwurfstätigkeit im eigentlichen Sinne, da sie weder inhaltliche Modellveränderungen leistet noch Voraussetzungen zum Entwerfen schafft oder Modellbewertungen erstellt. Sie umfaßt alle Prozesse der Informationstransformation und -interpretation, um eine koordinierte Abfolge von Entwurfsaktionen sowie deren benötigte Rückbezüglichkeit über Phasengrenzen zu ermöglichen.
In der Untersuchung nach WENZEL (Tab. 4-1) handelt es sich um die Tätigkeiten C – 'zeichnerische Dokumentation', G – 'Schriftwechsel' und H – 'Listen' mit ca. 25% Gesamtanteil in der frühen Entwurfsphase.
MATURANA [MAT87] bezeichnet Kommunikation als Verhaltenskoordination von Individuen oder Prozessen. Diese Definition setzt mindestens zwei Kommunikanten voraus, was die Rolle der Kommunikation im Entwurfsprozeß beleuchtet. Sie ist immer dann notwendig, wenn Prozesse (z.B. getragen von CAD-Tools) oder Personen mit verschiedenen deskriptiven Modellen (bei Personen a priori gegeben) am Entwurfs- und Herstellungsvorgang beteiligt sind. Kommunikation wird also an Phasengrenzen des sequentiellen, pragmatischen Entwurfsmodells notwendig.

***Change Model**:* Es werden formale Repräsentationen erzeugt, die wesentliche Modelleigenschaften, wie die Objektidentität, erhalten. Hierzu setzt allerdings dieser Aktionstyp eine ähnliche Modellierungstechnologie voraus. Die Verlustfreiheit der Modelltransformation ist aber auch hier nicht garantiert.
- Leite Statikmodell für 'Anbau' ab
- Erzeuge Eingangsdaten zur FEM-Berechnung 'Decke1_2'
- Erzeuge 3D-Volumen aus einem konstruktiven 2½D-Modell.

***Interprete Model**:* Dies ist Pendant zu *Change Model.* Die Verluste bei der Modellinterpretation resultieren aus den verschiedenen Modellabstraktionen, die aus einem unterschiedlichen Modellzweck herrühren sowie aus verschiedenen Domänenmodellen bei gleichem Modellzweck. Der Operationenverbund *Change/Interprete Model* wird in CAD-Systemen genutzt, wenn zwischen den Einzeltools die Kommunikation durch spezifische, modellbasierte Schnittstellen realisiert wird.
- Interpretiere Statikmodell für 'Anbau'
- Lese Geometriedaten zur FEM-Berechnung 'Decke1_2'
- Interpretiere STEP-File mit konstruktivem Modell von 'Anbau'.

Create Representation: Das Erzeugen informaler Repräsentationen (Bilder, Zeichnungen, Texte usw.) oder semiformaler Repräsentationen, wie genormte Schnittstellen (Ausschreibungstexte, DXF-Format usw.), wird mittels diesem Aktionstyp geleistet.
- Stelle Funktionsplan von 'Maiers_Haus' als Raumbuchlisting dar
- Stelle 'Maiers_Haus' als gerendertes Geometriemodell dar
- Erzeuge AVA-Texte für 'Anbau'.

Interprete Representation: Ist Pendant zu *Create Representation.* Es handelt sich um eine (stärker) verlustbehafte Form der Kommunikation. In der heutigen Praxis der CAD-Nutzung stellt sie die 'normale' Form der Kommunikation zwischen Personen und Prozessen dar. Sie kann von einer manuellen Wertübernahme auf Attributniveau, bis hin zu partiellen Rekonstruktion von Objektmodellen reichen (wenn die Schnittstelle hierzu geeignet ist und/oder die Kommunikaten den Modelltransfer standardisieren).

- Interpretiere DXF-Zeichnung für 'Maiers_Haus' als Geometriemodell
- Lese AVA-Texte für 'Anbau'.

***Comment**:* Bezeichnet das Assoziieren beliebiger (auch multimedialer) Dokumente mit Objekten oder Modellen, u.a. zur Verwaltung von Bürovorgängen.
- gesprochener Kommentar zum Bietergespräch Projekt 'Haus_Maier'
- Aktennotiz (Textdokument) zur Statiknachfrage 'Stütze27'

5.5 Steuerung

'Steuerungs'-Aktionen sind prozeßorientierte Aktionen, die sich mit der Organisation des Entwurfsablaufs befassen. Ihre Ausführung hat keine Modellveränderungen zur Folge. In formal-logischen Prozeßmodellen werden Steueraktionen nicht benötigt, da in ihnen einerseits das Entwurfsergebnis nicht von der Abfolge des Entwurfsaktionen abhängt und andererseits der Aufwand zum Erzeugen des deskriptiven Modells nicht betrachtet wird[4].
Im praktischen Entwurfsalltag spielt jedoch das Management von Ressourcen in Form privater oder kollektiver Arbeitszeiten sowie von technischen Ressourcen (z.B. Arbeitsplätze, CAD-Systeme) eine große Rolle. Terminliche Zwänge beeinflussen die Qualität von Entwurfsergebnissen beträchtlich und die Koordination der Ingenieurgewerke ist ebenfalls im Rahmen des Planungsprozesses zu leisten. In späteren Planungsphasen treten mit Projektplanung und -management handlungsbezogene Modelle in den Vordergrund.
Aussagen über den tatsächlichen Umfang organisierender Handlungsanteile im frühen Entwurfs lassen sich nur schwer gewinnen. WENZEL führt derartige Arbeitsanteile nicht auf. Aufwendungen für Überlegungen, wie 'Welche Teilaufgabe löse ich als nächstes?' werden nicht getrennt aufgeführt, sondern der selektierten Aufgabe zugeordnet. Ihr Aufwand dürfte jedoch beträchtlich sein, wie etwa der hohe Anteil entsprechenden Programmcodes in wissensbasierten Systemen zur Konfiguration zeigt.
Für CAD-Technologien entscheidet der Grad der angestrebten Unterstützung, in welchem Umfang Kontrolloperationen angeboten werden sollen. Ausgehend von der Programmierbarkeit, der Art des Modellbezugs und davon, wer auf Kontrolloperationen zurückgreifen kann, unterscheidet GÜNTHER ([GÜN92]):
- Strukturorientierte Kontrolle
- Datenorientierte Kontrolle
- Benutzerorientierte Kontrolle
- Fallorientierte Kontrolle
- Starre Kontrolle
- Programmierbare Kontrolle

Wichtigste Kontrollaktionen für die in frühen Phasen bevorzugten assistierenden CAD-Systeme ist der benutzerorientierte Kontrolltyp '*Undo*'.

***Undo**:* Rücknahme von modellorientierten Aktionen. Einfache Systeme gestatten die Rücknahme in der reversen Reihenfolge ihrer Anwendung ggf. mit einer beschränkten Rücknahmetiefe. Einfachste Variante wäre die komplette Speicherung alter Modellzustände. Weniger ressourcenintensiv ist die Speicherung inverser Operationen zu jeder Designaktion ([KOL94]). Aufwendigere Verfahren gestatten eine nichtchronologische, konsistente Rücknahme (gesteuertes Backtracking, ATMS/JTMS). Hierfür sind allerdings das Führen

[4] In der Prädikatenlogik 1. Stufe spielt beispielsweise die Stellung von Prädikaten in der Wissensbasis keine Rolle, sie kennt für die Verifikation einer Hypothese keine ablaufsteuernden Konstrukte. Für PROLOG als ihre technische Implementation gilt beides nicht.

einer tiefe gestaffelten Historie von Entwurfsaktionen sowie ein stark formalisiertes deskriptives und operationales Modell notwendig.

– Undo letzter Schritt, Undo bis ´erzeuge 'Bad1'´, Undo nur für ´erzeuge 'Anbau'´

***Set focus**:* Diese Kontrollaktion orientiert sich am Aufbau der Kompositionsstruktur. Sie basiert auf der Beobachtung, daß die Aufmerksamkeit des Entwerfenden auf einen Gegenstand bzw. ein Problem fokusiert ist, d.h. daß immer nur eine Entwurfsaufgabe aktiv vorangetrieben wird. Praktischer Nutzen ist, daß z.B. für Designaktionen nicht immer der vollständige Satz von Parametern explizit angegeben werden muß, sondern daß das fokusierte Objekt Gegenstand einer Sequenz von Entwurfsoperationen ist.

– Als nächstes bearbeite 'Etage1' **(neuer Fokus)**,
  verfeinere **(Fokus)** durch 'Flur', durch 'Bad', ...
– Setze 'Fläche' **(von Fokus)** auf 55 m^2.

Bemerkung: Der Zwang zur ständigen Aktualisierung des Fokus wird durch Nutzer oft störend empfunden, da diese Kontrolloperation mental unbewußt ohne Aufwand abläuft. Es sollte deshalb möglich sein, Entwurfsoperationen auch außerhalb des aktuellen Fokusses durch vollständige Parametrisierung auszuführen.

***Set task**:* Die explizite Deklaration von Entwurfstasks zielt auf eine Reduktion der Problemkomplexität. Dies gilt insbesondere auch für abgeleitete Repräsentationen, die dann gezielt nur Informationen der Task anzeigen. Da CAD notwendigerweise auf der externen Repräsentation des deskriptiven Modells erfolgt, zählen zu *set task* alle Arten des logischen Zoomings, da der 'Topknoten' der Aufbaustruktur, der als Repräsentant der Task dient, ein spezifisches Abstraktionsniveau besitzt. Für eine Task wird meist ein Analyse–Generiere–Evaluiere-Zyklus vollständig durchlaufen. Task können auch die Basis einer kooperativen Entwurfsorganisation bilden.

– Bearbeite heute 'Eingangsvariante'
– Bearbeiter 'Maier' bearbeitet 'Anbau'.

***Set context**:* Alle Repräsentationen dienen letztlich einem Kommunikationsprozeß. Kommunikation setzt immer eine Interpretation in einem Kontext voraus. Dies bezieht sich sowohl auf eine informale Kommunikation von Personen als auch auf eine formale Kommunikation zwischen CAD-Prozessen (etwa in einem computergestützten Evaluierungsprozeß). Da das den Kontext bildende Domänenwissen dynamisch ist, können Kontexte sowohl Zeitbezogenheit als auch Personenbezogenheit ausdrücken. Darüber hinaus dienen Kontexte auch zur Reduktion des Suchaufwandes etwa bei Analyseoperationen durch Ausschnittsbildung innerhalb einer spezifischen Wissensdomäne. Modelländerung, beispielsweise im Zusammenhang mit der kooperativen Bearbeitung innerhalb von Ingenieurgewerken, sollte allerdings durch den Aufbau eines neuen Modells im Rahmen eines Kommunikationsprozesses erfolgen.

– Kontext für Task 'Anbau' ist 'öffentliche Gebäude' *(Domänenbezogenheit)*
– Kontext für Task 'Anbau' ist 'Hotelbauten' *(Ausschnittsbildung)*
– Kontext für Task 'Anbau' ist Wissensbasis vom '1.6.96' *(Zeitbezogenheit)*
– Kontext für Task 'Anbau' ist Bearbeiter 'B' *(Subjektbezogenheit)*

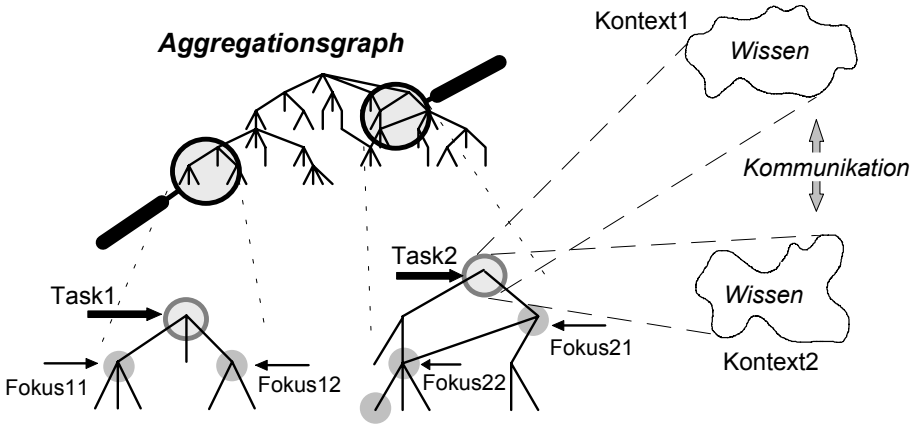


Abb. 5-5 Entwurfssteuerung mit Kontext, Task und Fokus

***Set strategy**:* Hier handelt es sich um Aktionen, die für CAD Systeme der Kategorie 'design automat' erforderlich sind. Strategien müssen als formalisierte Handlungsanweisung vorliegen. Sie erfordern deshalb ein stark formalisiertes Produkt-/Domänenmodell. Es sind aber auch Verweise auf informale Handlungsanweisungen denkbar.

- Verwende Generiere–Teste-Strategie für 'Dachstuhl', generiere Formelemente aus Kontext Ziegeldach, selektiere Lösung nach minimalen Kosten
- Verwende für Task 'Keller' 'least commitment' Strategie[5]
- Löse Problem 'Treppe Erdgeschoß' nach Verfahren aus 'NEUFERT' (informal)

***Select**:* Sowohl mit einer Designstrategie als auch ohne eine solche, ist eine Auswahl des nächsten zu bearbeitenden Problems (Task oder Fokus) von Interesse. Das Auswahlkriterium kann durch die Designstrategie implizit oder durch den Nutzer explizit vorgegeben sein. Die Suchen des nächsten zu bearbeitenden Objektes wird wesentlich von einer Bewertungsfunktion für Entwurfsaktionen bestimmt. Solch Bewertungsfunktionen könnten etwa sein: benötigte Ressourcen für die Problemlösung, Ergebnis des Evaluierungsprozesses, Grad des Entwurfsfortschritts für die jeweilige Task, Grad der Verletzung von Constraints u.a.m.

- Bei welchem Objekt sind die meisten Constraints verletzt ?
- Welches Teilproblem hat die größten Kosten?
- Welche Probleme müssen heute noch gelöst werden ?

***Propagate**:* Dieser Kontrolltyp setzt ein stark formalisiertes Entwurfsmodell voraus. Er befaßt sich mit der (automatischen) Fortpflanzung von Modellveränderung sowie der Behebung von Inkonsistenzen, die durch Designaktionen (i.allg. Analyse oder Synthese) induziert werden. Dieser Kontrolltyp kann Teil einer umfassenden Designstrategie sein, wie etwa der 'least commitment' Strategie. In technischen Systemen wird Propagation durch Regeln (unidirektionale Ausbreitung, vorwärtsverkettende Regeln) oder durch Constrains (multidirektionale Ausbreitung, siehe [CUN91]) realisiert.

- Ändere Entwurfsparameter, die als Folge der Änderung der Traufhöhe inkonsistent sind
- Verändere alle erforderlichen Entwurfsparameter als Folge der Änderung der zu erzielenden Brandschutzklasse
- Generiere alle Teile, die als Folge der Entwurfsentscheidung 'Solarheizung' nötig werden.

[5] Strategie, nach der der Bereich für einen Wert (mögliche Belegung) bei Parametern durch Generieren und Propagieren von Constraints sukzessive eingeschränkt wird. (Für diese und weitere Strategien siehe [PUP91]).

5.6 Zusammenfassung

Die Entwicklung eines generischen Prozeßmodells, das für alle Phasen des Bauwerksentwurfs gleichermaßen verbindlich sein kann, ist gegenwärtig nicht zu erwarten. Im Kontrast zu gegenständlichen Modellen sind Prozeßmodelle gegenwärtig nur in geringem Maße Thema von Forschungsarbeiten. Die vorgeschlagenen Basistätigkeiten nehmen die Erkenntnisse aus kognitiven Untersuchungen zum 'Wie' des Entwurfs auf und setzen sie um. Für ihre umfassende praktische Nutzung ist noch eine intensive Diskussion erforderlich. Diese Diskussion muß einerseits mit Architekten geführt werden, inwieweit es ihnen gelingt, relevante Entwurfsaufgaben mit der angebotenen Grundmenge von Operationen zu lösen. Andererseits muß mit Softwareentwicklern geklärt werden, ob sich die atomaren Entwurfsaktivitäten zur phasenübergreifenden Organisation von CAD-Prozessen eignen.

6. OBJEKTORIENTIERTE MODELLE ALS AUSDRUCKSMITTEL IM ENTWURF

6.1 Modellbildungsprozesse

Geht man von der angeführten Definition des Entwurfsbegriffes nach SALZMANN (Kapitel 4) aus, so ist das Ziel dieses Prozesses eine Vorabbildung baulicher Objekte. Folgt man dem repräsentationalistischen Ansatz, erfolgt dieses 'Vorausdenken' mit Hilfe gedanklicher Widerspieglungen von potentiell beobachtbaren *Objekten*. Diese Objekte, ihr Zustand und ihre Beziehungen untereinander konstatieren in ihrer Gesamtheit ein Modell des Bauwerkes so wie es ist oder sein sollte. Bei der Definition des Begriffs *Modell* legt STACHOWIAK [STA73] Merkmale fest, die für das Modell als Ganzes gelten, aber auch rekursiv für die abgrenzbaren Elemente, die das Modell bilden. Diese Merkmale sind das

- *Abbildungsmerkmal*
  Modelle bilden etwas ab, d.h. es gibt für sie reale oder künstliche Originale[1], deren Entsprechung sie sind.
- *Verkürzungsmerkmal*
  Modelle sind kein vollständiges Abbild des Originalsystems. Sie verkürzen Modellaspekte gemäß eines Modellzwecks auf relevante Eigenschaften des Originals.
- *Subjektivitätsmerkmal*
  Modellbildung als Abbildungsprozeß ist eine Tätigkeit, die subjektgebunden ist. Das Modell friert die Art der Abbildung auf die Repräsentationsbedürfnisse des Subjektes zum Zeitpunkt der Modellbildung ein.

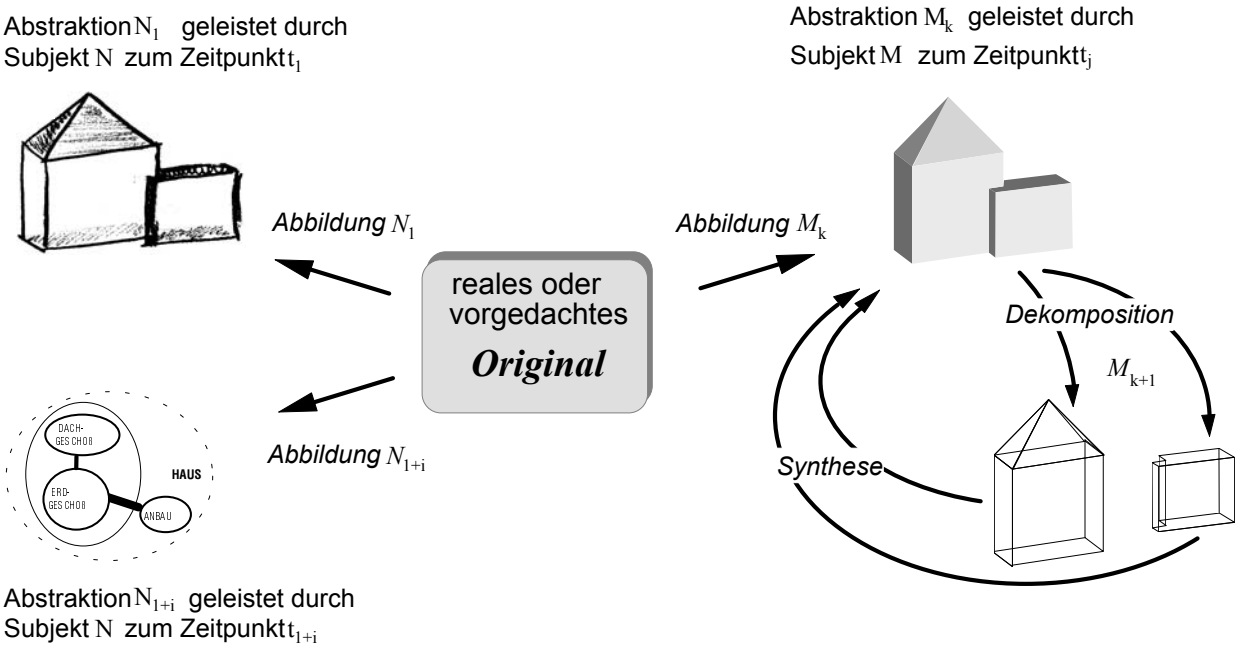


Abb. 6-1 Modelleigenschaften: Abbildung, Abstraktion, Subjektivität

Doch worin liegt die Bedeutung der Modellbildung? Der Prozeß des 'Vorausdenkens' dieser Bauwerke im Planungsprozeß dient dem Zweck, Prognosen oder Zusagen ihrer Nutzungsfähigkeit und Nutzungsökonomie zu treffen. Dies kann nicht experimentell nach der

[1] DÖRNER [DÖR91a] benutzt treffender das deutsche Wort 'Urbilder'

trial-and-error Methode erfolgen[2]. Die erforderlichen Prognosen werden anhand von Modellen getroffen, die die Realität oder eine erwartete Realität abbilden (Abbildungsmerkmal). Zur Beherrschung dieser Komplexität setzt der Mensch drei Techniken ein: *Abstraktion*, *Dekomposition und Kooperation*.

- Die *Abstraktion* ermöglicht die informationelle Verkürzung von Modellen soweit, daß zweckmäßige, überschaubare Entwurfshandlungen auf diesen Modellen möglich werden. Das Spektrum möglicher Handlungen reicht vom bewußten Räsonieren bis hin zu einer unbewußten Intuition, die Modellzustände holistisch rezipiert. So kann etwa der Abstraktionsprozeß bewußt so gestaltet werden, daß auf dem Modell z.B. bekannte Berechnungsverfahren angewandt werden können. Er kann sich aber auch unbewußt vollziehen, so daß in einem sozialen Kontext (etwa einer Diskussion) möglichst gleichartige Modellinterpretationen erzielt werden. Der Grad der Abstraktion variiert also mit dem jeweiligen Modellzweck und mit der Komplexität des Problem- und des Lösungszustandes.

- Die *Dekomposition* strukturiert Entwurfsprobleme und damit die zugehörigen Modelle, die zu ihrer Lösung entwickelt wurden. Sie ist Teil der Designstrategie (siehe Abs. 4.2.2) Ein komplexes Anfangsmodell wird in weniger komplexe Teilmodelle dekomponiert. Die Tiefe bis zu der die Modelldekomposition getrieben wird, hängt vom Modellzweck und dem zugehörigen Abstraktionsgrad ab. So terminiert Modellaufspaltung für die frühe Konzeptphase von Einfamilienhäusern auf Raumniveau mit dem zu dieser Art von Objekten gehörenden Abstraktionsgrad. Ausführungsunterlagen, die in späteren Entwurfsphasen erarbeitet werden, müssen sehr viel präziser sein.

- Die *Kooperation* erfordert in jedem Fall eine Externalisierung mentaler Modelle, um sicherzustellen, daß alle an der Kooperation Beteiligten am selben Problem arbeiten. Basis der Kooperation sind das Erzeugen sowie das Interpretieren von Abbildungen, um eine möglichst gute Koordination der Kommunikanten zu erreichen[3]. Zur Organisation von Kooperation bieten sich grundsätzlich zwei Wege an:

  – sequentielle Kooperation durch spezialisierte Agenten
    In diesem Fall werden wiederum verkürzte Modelle benötigt, wobei die Art der Abstraktion durch den jeweiligen Modellzweck des Spezialisten bestimmt wird.
  – parallele Kooperation durch gleichartige Agenten
    Hier erfolgt eine Dekomposition in weitgehend eigenständige Teilmodelle/ Teilprobleme. Je nach dem Grad der Separierbarkeit der Teilmodelle ist die Synthesevorschrift zur Koordination der parallelen Bearbeitung von essentieller Bedeutung.

Alle bisher genannten Eigenschaften der Modellbildung zielen auf die Abbildung eines singulären, beobachtbaren oder zumindest potentiell beobachtbaren Objektes. Im Gegensatz dazu stehen Verfahren der Modellbildung über Diskursbereiche oder Domänen. Ziel ist hier das Erstellen generalisierter Modelle, die Mengen potentieller Ausprägungen beschreiben. Das Abbildungsprinzip ist hier nicht die Abstraktion, sondern die *Generalisation* (siehe Abb. 6-2).

[2] dazu RITTEL [RIT92]: „Wann immer Handlungen tatsächlich irreversibel sind und die Halbwertszeiten der Konsequenzen lang sind, zählt jeder Versuch“

[3] MATURANA [MAT87] bezeichnet nicht den *Austausch* von Informationen (Röhrenmetapher) als Kommunikation, sondern die *Verhaltenskoordination* der kommunizierenden Einheiten.

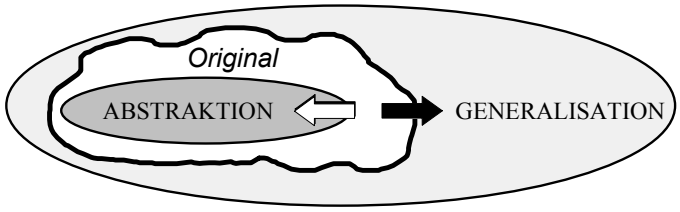


Abb. 6-2 Verhältnis *Abstraktion - Generalisation*

Die Verfügbarkeit generalisierter Modelle ermöglicht es allgemeingültige Aussagen, Lehrsätze oder Verfahren zu formulieren, die in ihnen bzw. für sie gelten. Deren Anwendung erbringt dann für konkrete Ausprägungen des generalisierten Modells die erwünschten Ergebnisse, etwa in Form von Modellerweiterungen (Produktion) oder von Modellverifikation bzw. -validation (Resolution/Deduktion). Diese Handlungen werden oft als 'Räsonieren', als einer bewußten Form geistiger Tätigkeit[4] bezeichnet. Sie sind Gegenstand von Lehrbuchwissen und von Verfahren (Algorithmen) , die zur Gestaltung von CAD-Tools geeignet sind.

Die ambivalente Nutzung des Modellbegriffs führt zu grundlegenden Verständnissproblemen. Gegenstand vieler Veröffentlichungen ist die *Produktmodellierung.* Dies bezieht sich auf die Erstellung generalisierter Modelle. So werden z.B. 'B-Rep'-Modelle als Geometriemodelle bezeichnet. Es handelt sich jedoch um eine verallgemeinerte Vorschrift, mit deren Hilfe abstrahierende Abbildungen konkreter Geometrien erzeugt werden können. Bei der Anwendung jeglicher 'data definition languages' (DDL) wie etwa EXPRESS, oder entsprechender grafischer Notationen, wie die nach RUMBAUGH ([RUM91]), wird kein *einzelnes* Produkt modelliert, sondern eine *Menge* möglicher Produkte. Will man Verfahren/ Programme anwenden, die auf dieser Menge agieren können, muß ein konkretes Produkt modelliert sein. Im Rahmen dieser Arbeit wird darum stattdessen der Begriff *Domänenmodelle* verwendet (siehe Glossar im Anhang).

Im Rahmen der CAD-Technologie werden gegenwärtig Generalisationsprozesse lediglich von Softwareentwicklern geleistet. Anwender können generalisierte Modelle weder verändern noch erweitern. Sie können im allg. auf die Modelle, die der Software zugrundeliegen, nicht einmal lesend zugreifen. Abstraktionsoperationen auf Elementen des Bauwerksmodells sind im Zuge des Entwurfsprozesses ebenfalls nicht möglich, da deren Abstraktionsgrad von der Art der Generalisierung des Konzeptes bestimmt wird, von dem sie abgeleitet wurden. Eine Variation des Abstraktionsgrades von Modellelementen durch den Nutzer wird nicht angeboten. So ist es nicht möglich, ein Raumobjekt anfangs als 'Naßraum' zu qualifizieren (zu klassifizieren) und ihn im Zuge des Entwurfsprozesses später zur 'Dusche' zu spezialisieren oder umgekehrt.

6.2 Modellelemente – das 'Material' der Modellbildung

Für CAD-Tools, die jene Entwurfsphasen unterstützen, in den ein dynamisches und/oder subjektives Domänenmodell erforderlich ist, müssen neben den abstrahierenden Elementen auch generalisierte Modellelemente darstellbar sein. Dabei stehen beide Modellierungsverfahren in engem Zusammenhang. Beim modellierenden Planen werden aus dem Vorrat an generalisierten Modellelementen die ausgewählt, die die Intensionen des Planers am besten

[4] DREYFUS/ DREYFUS [DRE91] unterscheiden vom Novizen bis zum Expertentum 5 Level von Handlungskompetenz (dort Expertise genannt). Räsonieren, d.h. daß befolgen bewußter Handlungskonzepte führt maximal bis Level 3 der Kompetenz und ist damit nicht Gegenstand wirklichen Expertentums.

wiedergeben. Durch die Selektion und anschließende Anpassung verlieren sie den Charakter generalisierter Elemente und werden nun zu abstrahierenden, aber konkreten Abbildungen des Artefakts.

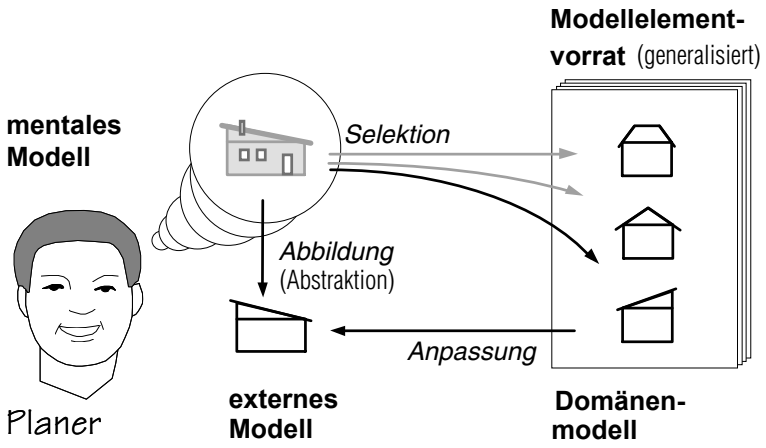


Abb. 6-3 Selektion und Anpassung generalisierter Modellelemente

Die Darstellung von generalisierten und abstrahierenden Elementen erfolgt extern in Symbolen, die als Assoziationen für Modellvorstellungen beider Typen fungieren. Diese Technik der expliziten Modellbildung geht also vom repräsentationalistischen Ansatz aus. In der Softwaretechnik folgt das Paradigma der *Objektorientierung* dem beschriebenen Vorgehen. Besonders der geforderte Zusammenhang zwischen generalisierten und konkreten Modellelementen ist hier durch das Konzept der Klassen und Instanzen gegeben.

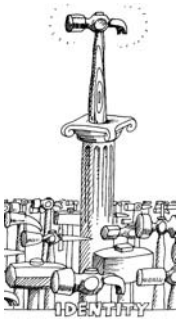
Gegenwärtig gibt es in der Praxis keine Programme, die diese Modellierungstechnologie auch zur Bildung von Bauwerksmodellen durch den Entwerfenden anbietet. In der Softwaretechnik ist die Zielrichtung dieser Technologie die Organisation von Programmen und deren Entwicklungsprozesse. In diesem Kontext besitzen die Elemente des Modellierungsparadigmas eine spezifische Semantik, die sich von der unterscheidet, die aus einer Nutzung zur Modellbildung in der Planung resultiert.

Auch wenn das OO-Paradigma gut bekannt und beschrieben ist ([RUM91], [BRE92], [MAR94] u.v.a.) werden die Elemente dieser Modellbildungstechnik einer eingehenden Untersuchung unterzogen, um ihre Eignung für den Planungsprozeß zu klären. Die jeweils spezifische Semantik wird dabei immer wieder gegenübergestellt.

6.2.1 Objekte – Modellhafte Abbildungen realer Entitäten

Objekte bezeichnen abgrenzbare Entitäten oder Sachverhalte. Die menschliche Fähigkeit, solche Objekte zu finden/zu erkennen und sie zu benennen/zu repräsentieren ist Grundlage unseres kognitiven und intelligenten Agierens. MATURANA / VARELA formulieren:
„Das Aufzeigen eines Wesens, Objektes, einer Einheit oder Sache ist mit einem Akt der Unterscheidung verbunden, der das Aufgezeigte von einem Hintergrund unterscheidet und damit von diesem trennt. ... Es ist eine ganz alltägliche Sache und nicht etwa eine besondere Situation, in der wir uns andauernd und notwendigerweise befinden.“ und weiter „Eine Einheit (Wesen, Entität, Objekt) ist durch einen Akt der Unterscheidung definiert. Anders herum: Immer dann, wenn wir in unseren Beschreibungen auf eine Einheit Bezug nehmen, implizieren wir eine Operation der Unterscheidung, die die Einheit definiert und möglich macht.“ ([MAT87])

Durch diesen Akt der Unterscheidung wird für Objekte die Zuordnung von Eigenschaften erst möglich, wie sie BOOCH in [BOO94] fordert und dort zur Basis eines technischen Konzeptes zur Softwareproduktion macht.



Identität	Zustand	Beziehung	Verhalten
Jedes Objekt verfügt über Eigenschaften, die es in der Welt einmalig machen. Die Menge der Merkmale, die diese Einmaligkeit begründen, heißt Identität.	Die Gesamtheit der (üblicherweise statischen) Eigenschaften des Objektes mitsamt der (üblicherweise dynamischen) Eigenschaftswerte bilden dessen Zustand.	Die Beziehung zwischen Objekten umfaßt die Annahmen, die jedes über die anderen hat sowie Operationen und resultierendes Verhalten, das auf der Beziehung basiert.	Das Verhalten beschreibt die dynamischen Veränderungen und Reaktionen eines Objektes auf Zustandsänderungen der Außenwelt, die ihm in Form von Nachrichten übermittelt werden.

Tab. 6-1 Objekte und deren Merkmale nach [BOO94]

Die Literatur zur OO-Softwareentwicklung gibt übrigens wenig Hinweise darauf, *wie* Objekte gefunden werden können. Dies ist ein Mangel, der häufig beklagt wird. Hier zeigt sich, daß das Finden von Objekten ein kognitiver Akt ist, der keineswegs trivial ist.
Im eigentlichen Schaffensprozeß agiert der Architekt nicht mit Elementen der Repräsentationen wie Worten, Buchstaben, Zahlen, Symbolen, Linien. Die genannten Elemente sind lediglich mit Objekten assoziiert, die in der Realität existieren oder in seiner Imagination geschaffen werden. In seiner Vorstellung sind die Objekte jedoch viel reicher als ihre externen Repräsentation. Repräsentationsformen werden entweder von einzelnen oder kleinen Gruppen von Personen formal entwickelt (B-Rep, STEP) oder sie werden in einem langwierigen sozialen Prozeß gebildet (natürliche Sprachen). Genau diese ist gemeint, wenn STACHOWIAK vom Subjektivitätsmerkmal der Modellbildung spricht. Die Repräsentationsformen mußten einem Prozeß der Konsensfindung unterliegen, um ihre Interpretation zu normieren und so Kommunikation und damit Kooperation zu ermöglichen (Abb. 6-4).

Die Identität von Modellobjekten ist nicht an spezielle Entwurfs- oder Bauwerkslebensphasen gebunden. Sie können über Phasengrenzen hinweg existieren. Ihre Identität bleibt bestehen, auch wenn sie geändert, anders interpretiert oder präsentiert werden und gelegentlich auch ihr Abstraktionsgrad entsprechend der Erfordernisse der Bauwerkslebensphase modifiziert wird.

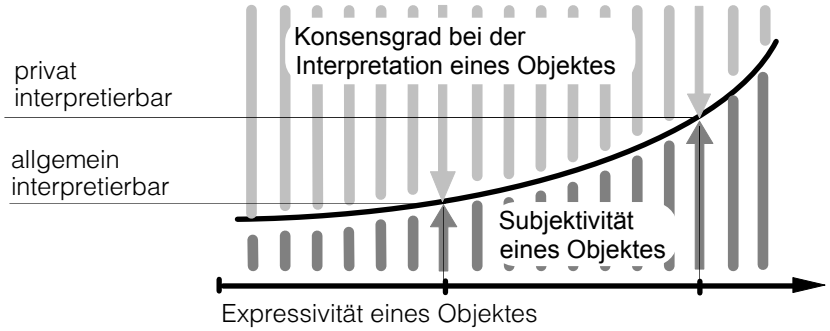


Abb. 6-4 Subjektivitätsmerkmal von Modellabbildungen

Identität ist ein Merkmal, das in der Realität ganz natürlich existiert, das aber in einer computergestützten objektorientierten Modellierumgebung erst hergestellt werden muß.

Der im folgenden verwandte Objektbegriff unterscheidet sich deutlich von dem in der Objektorientierten Programmierung (OOP) oder dem Objektorientierten Softwaredesign (OOD). So bezeichnet RUMBAUGH ([RUM91]) ein Objekt in diesem Bereich als „einfach etwas, das im Kontext einer *Applikation* Sinn macht“ und das zwei Zwecken dient: „dem Verständnis der realen Welt und als praktische Basis für eine Computerimplementation“.

Eine CAD-Entwurfsumgebung muß zum Modellieren Objekte bereitstellen, die Reflektionen von Elementen der Imagination oder der Realität sind. Die Objekte müssen über eine Identität verfügen, die sie im späteren Gebäude auffindbar macht. Ihre Bedeutung ergibt sich, entgegen RUMBAUGH, aus dem Modellierungskontext, in dem sie verwendet werden. Die rechnerinternen Objekte in einem CAD-System, die über eine stoffliche Entsprechung in der realen oder imaginierten Welt verfügen, werden als *'real world objects'* bezeichnet (Abb. 6-5).

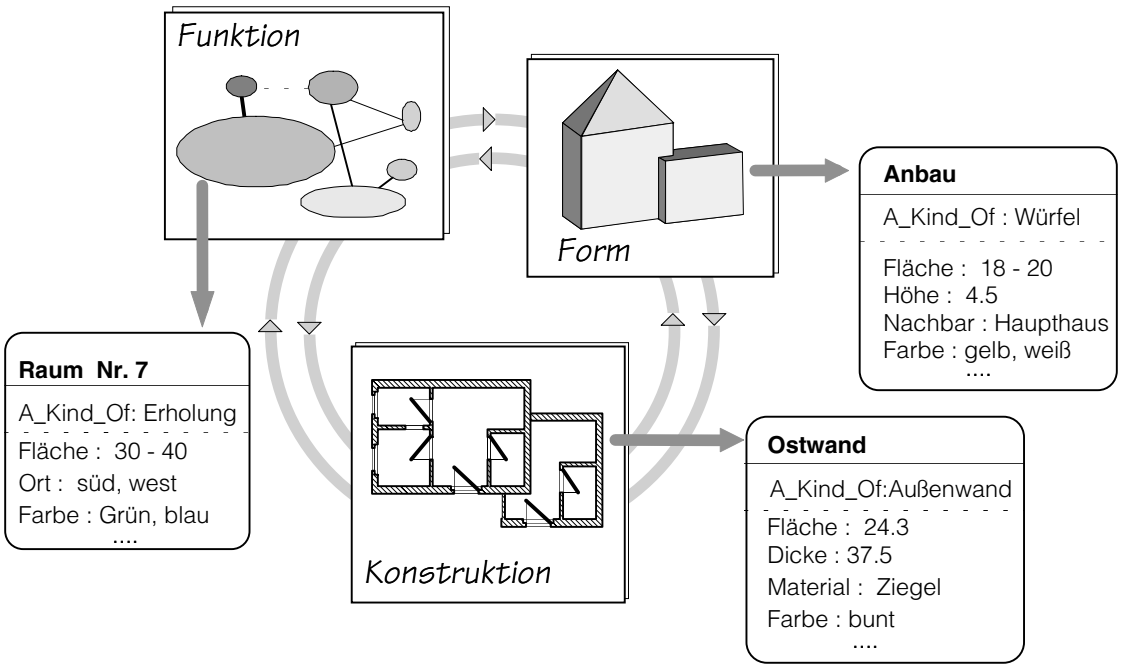


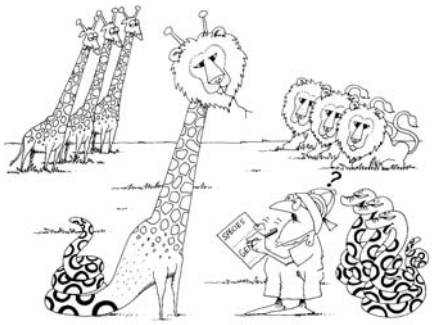
Abb. 6-5 *'real world' Objekte*

6.2.2 Klassen – Generalisation von Objekten

Durch die Generalisierung singulärer Objekte wird ein abstraktes Objekt gebildet, das eine Menge von Ausprägungen repräsentiert. Dieses *Klassenobjekt* ist letztlich die Definition einer inneren Organisation der Menge von Objekten. Die Wahrnehmung dieser inneren Struktur bei einem Objekt gestattet uns seine Zuordnung zur abstrakten Klasse. MATURANA/VARELA führen hierzu aus:

> „Es sind solche Relationen, die existieren oder gegeben sein müssen, damit ein Etwas etwas ist. Damit ich ein Objekt als einen Stuhl bezeichnen kann, muß ich zuvor anerkennen, daß gewisse Relationen zwischen den Teilen, die ich Beine, Lehne, Sitzfläche nenne, auf eine Weise gegeben sind, die das Sitzen möglich machen“ und weiter „Die Tatsache, daß wir beim Aufzeigen oder Unterscheiden eines Objektes implizit oder explizit dessen Organisation anerkennen, ist insofern universell, als wir diesen Akt andauernd als einen grundlegenden kognitiven Akt vollziehen. Er besteht in nicht weniger und nicht mehr als dem Erzeugen von Klassen jedweder Art.“

Klassenobjekte beschreiben neben den Ausprägungen, von denen sie abgeleitet wurden, auch zukünftige Ausprägungen. So wird es möglich, durch die Zuordnung eines beobachteten oder imaginierten Objekts zu einer Klasse Annahmen[5] über den Aufbau und die Wirkungsweise des Objekts zu treffen. Man bedient sich hierzu der induktiven Schlußweise, die die erwiesene Gültigkeit von Gesetzmäßigkeiten bei einer genügend großen Menge von gleich strukturierten Einzelfällen auf alle gleich strukturierten Fälle ausdehnt. Mit Hilfe der Klassen als Generalisierung wird es also möglich, Wissen an diese Klasse zu binden. Dies ist eine ganz wesentliche Voraussetzung des Lernens. Man lernt nicht genau diese Treppe an genau jener Stelle zu bauen, sondern man lernt das Bauen von Treppen, d.h. man erwirbt Wissen, das zur Klasse der *Treppen* gehört.



Eine Klasse ist eine Verallgemeinerung von (konkreten) Instanzen oder (präziseren) Klassen durch Beschreibung von gemeinsamen Eigenschaften und Verhalten [BOO94].

Abb. 6-6 Klassifikation nach BOOCH

Wenn eine Klassendefinition auf der Struktur und den Eigenschaften von Individuen einer Objektmenge beruht, handelt es sich bei der Klasse um eine implizit gebildete Menge, d.h. diese Menge ist nicht durch die Aufzählung ihrer Mitglieder (Instanzen) gegeben.

Ein Klasse K wird gebildet für reale oder virtuelle (n - bekannte, m - zukünftige) Individuen o_i derart, daß

für $K = \{o_i\}$ ein Prädikat gilt mit $\forall o_i | p_K(o_i)$ $\quad i = 1...n...(n+m) \quad n, m \in N$

In diesem Sinne sind Menge und Klasse analoge Begriffe.

[5] REIMER ([REI91]) unterscheidet Wissen, wenn die Annahmen tatsächlich wahr sind und Überzeugungen, wenn deren Wahrheitsgehalt nicht feststeht.

Der beschriebene Ansatz wird als '*Classical Categorization*' bezeichnet. Das '*Conceptuel Clustering*' beachtet die Tatsache, daß oft kein Satz objektiver Merkmale zur Generalisation vorliegt, d.h. ein Prädikat p_K ist nicht explizit gegeben, sondern die Zuordnung eines Objektes o_i zu einer Klasse K wird pragmatisch getroffen. Die '*Prototyp Theory*' betont dagegen die Analogie. Ein häufig beobachtetes Objekt o_{prot} dient als Prototyp der Klasse K. Neue Instanzen o_j werden als Kopien des Prototypen o_{prot} erzeugt. Die Zugehörigkeit eines neu beobachteten Objekts o_k kann ähnlich der 'fuzzy set theory' über eine Abstandsfunktion f_{abst} festgestellt werden, es gilt:

$$o_k \in K \quad \text{wenn} \quad f_{abst}\left(o_{prot}, o_k\right) \leq \varepsilon_K, \qquad \varepsilon_K \text{ - Schranke}$$

In der Programmierungstechnik ist der Begriff der Klasse (auch historisch) vom Begriff des Typs abgeleitet. Zwischen dem vorgestellten Klassenbegriff und dem des Typs gibt es jedoch beträchtliche inhaltliche Differenzen. Programmiersprachliche Typen sind Vorschriften zur Strukturierung von Speicherbereichen, die zur Laufzeit Instanzen des Typs aufnehmen können. Der Typ selbst ist auf unterster Ebene ein Komposit aus programmiersprachlichen Basistypen. Zur Typbildung als Generalisierung dient die statische Menge der einem Typ zugeordneten Merkmale und deren Verhalten (Methoden). Der Ordnungs- und Kapselungsaspekt steht bei diesem Vorgehen im Vordergrund. Die Typbildung wird durch Softwareentwickler vollzogen.

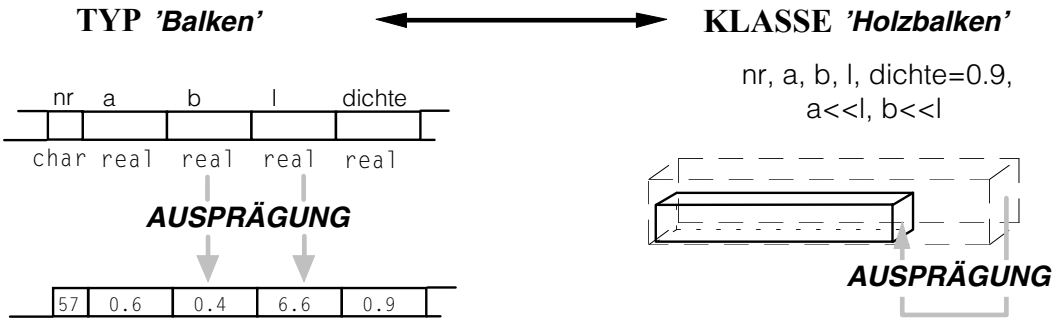


Abb. 6-7 Typen und Klassen

Die Architekturtheorie faßt den Klassenbegiff noch weiter und spricht vom Typus oder von Pattern. ALEXANDER beschreibt Entwurfshandlungen des Architekten als gänzlich bestimmt von „pattern he has in mind at that moment, and his ability to combine these patterns to form an new design" ([ALE79]). AKIN ([AKI88]) spricht vom 'Architekturtypus' und BROADBENT ([BRO73]) vom 'Ikonischen Entwurf'[6] . Beiden Sichten ist gemeinsam, daß konkrete Entwurfselemente (Entwurfs-Objekte) von einem abstrakten Typus oder eben der Ikone abgeleitet werden. Der Typus verfügt über Eigenschaften, die ihn prototypisch an Stelle eines konkreten Elementes wirken lassen können. Erkenntnistheoretisch ist zu unterscheiden in 'Erkennen', als dem Prozeß des Trennens von einem Hintergrund und dem 'Wiedererkennen' als dem Prozeß des Zuordnens zu einem abstrakten, summarischen Typus, d.h. dem Prozeß des Klassifizierens.

[6] BROADBENT beschreibt 'Iconic Design' als Nutzung erprobter und allgemein akzeptierter Formen und Konzepte, die als prototypische Lösungen fungieren. Als weitere Varianten des Design Prozesses nennt er 'Pragmatic Design', 'Analogical Design' und 'Canonic Design'.

In unserer durch Deduktion und Symbolmanipulation geprägten Denkwelt werden solche Klassen in Symbolen repräsentiert. Symbole sind die lesbare Assoziation für Klassen. Auch wenn hiermit nur ein kleiner Teil unserer Erkenntnisfähigkeit repräsentiert wird (vergl. [MAT87], [DRE91]), lehnen sie sich doch gut an die von Erkenntnisspsychologen erarbeitete Theorie der 'knowledge chunks' an (vergl. [GLO91], [DRE91], [AKI86]). Im Rahmen von Wissensrepräsentation der Künstlichen Intelligenz erfahren sie ihre konzeptuelle Umsetzung in FRAMES ([MIN75]) und ihre technische Realisierung in 'Frame Representation Languages' wie z.B. FRL in [STO91]. Letztere sind mit OO-Programiersprachen verwandt, aber konzeptuell wesentlich mächtiger.

Entwerfen als Prozeß der Erstellung des Modells eines Artefaktes bedeutet im Rahmen des ikonischen Entwurfs solche abstrakten Inkarnationen eines architektonischen Typus in den konkreten Entwurfskontext zu stellen und so zu adaptieren, daß sie diesem Kontext angemessen sind. Im Sinne der Softwaretechnologie handelt es sich dabei um Instanziieren.

6.2.3 Taxonomien – Ordnung der Klassen

Um die zur Beschreibung oder Behandlung eines bestimmten Sachverhaltes richtige Klasse zu finden, müssen diese geeignet geordnet sein. In einer CAD-Entwurfsumgebung sollte das Ordnungsverfahren dem Nutzer vertraut und ggf. wechselbar sein.

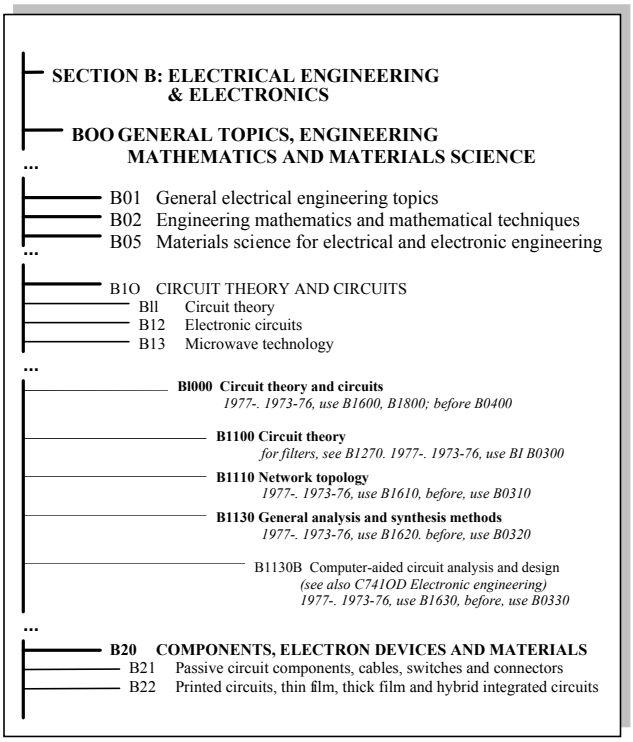
Repräsentationsorientierte Ordnungen sind eine mögliche Ordnungsmethodik. Mögliche Ordnungskriterien sind z.B. Adressen (etwa im Hauptspeicher), Indizes, zeitliche Reihenfolge der Erstellung oder Merkmale der Repräsentation selbst (z.B. alphabetische Abfolge von Bezeichnern).

Inhaltsorientierte Ordnungen ordnen Konzepte nach ihrer Bedeutung in einem Kontext oder für ein Subjekt. Eine häufige Form der Darstellung solcher Ordnungssysteme sind Graphen, deren Knoten die Konzepte/Klassen sind, während die Kanten deren semantische Relationen abbilden. Ordnungssysteme sind wichtiger Forschungsgegenstand innerhalb der Künstlichen Intelligenz (siehe u.a. semantische Netze[7], Abb. 6-8 b), in der Linguistik, für Informationssysteme und im Bibliothekswesen (Thesauri, siehe Abb. 6-8 a).

Eine häufige Beziehung zwischen Klassen ist, daß eine geeignete Bedingung existiert, mit der sich in einer Klasse K_i (Superklasse) eine Teilmenge von Dingen unterscheiden läßt. Diese Teilmenge bildet eine Subklasse K_{ij} innerhalb K_i, sie stellt eine Präzisierung oder Spezialisierung der Superklasse K_i dar. Wird der Prozeß der Spezialisierung für die Subklassen rekursiv wiederholt, erhält man ein hierarchisch gegliedertes System von Mengen. Dessen Darstellung als Graph wird als **Taxonomie** bezeichnet.

Diese Art der inhaltsorientierten Strukturierung von Objektsystemen baut auf dem gleichen Prinzip der Unterscheidbarkeit von Objekten und Konzepten auf, wie die Generalisierung von Instanzen zu Klassen.

[7] von QUILLAN 1969 erstmals umfassend vorgestellt ([STO91])



SECTION B: ELECTRICAL ENGINEERING & ELECTRONICS

BOO GENERAL TOPICS, ENGINEERING MATHEMATICS AND MATERIALS SCIENCE

...

B01 General electrical engineering topics
B02 Engineering mathematics and mathematical techniques
B05 Materials science for electrical and electronic engineering

...

B1O CIRCUIT THEORY AND CIRCUITS
Bll Circuit theory
B12 Electronic circuits
B13 Microwave technology

...

Bl000 Circuit theory and circuits
1977-. 1973-76, use B1600, B1800; before B0400

B1100 Circuit theory
for filters, see B1270. 1977-. 1973-76, use BI B0300

B1110 Network topology
1977-. 1973-76, use B1610, before, use B0310

B1130 General analysis and synthesis methods
1977-. 1973-76, use B1620. before, use B0320

B1130B Computer-aided circuit analysis and design
(see also C741OD Electronic engineering)
1977-. 1973-76, use B1630, before, use B0330

...

B20 COMPONENTS, ELECTRON DEVICES AND MATERIALS
B21 Passive circuit components, cables, switches and connectors
B22 Printed circuits, thin film, thick film and hybrid integrated circuits

Abb. 6-8 a Auszug aus dem Thesaurus 'INSPEC CLASSIFICATION' ([INS88]) der amerikanischen Elektroingenieure

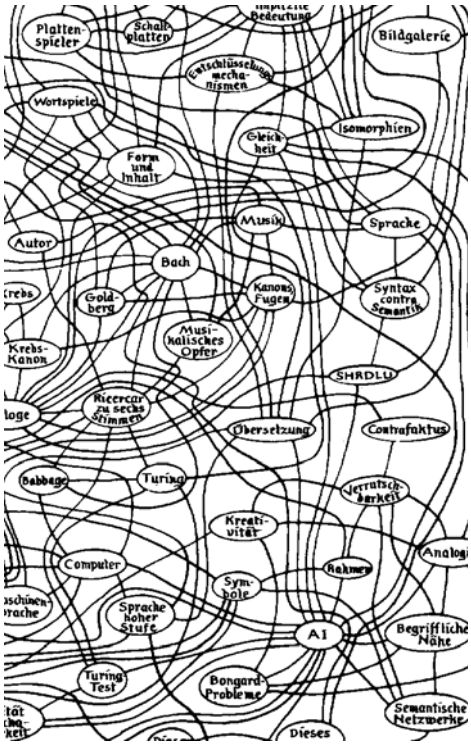


Abb. 6-8 b „ein winziger Ausschnitt des 'semantischen Netzwerkes' des Autors ...", nach HOFESTADTER [HOF92]

Für Bildung einer Superklassen Ks aus Subklassen K_i gilt, daß

$$Ks = \left\{ \bigcup_{i=1}^{n} K_i \cup \{o\} \right\} \text{ mit } \left(\forall o | p_{Ks}(o)\right) \wedge \left(\forall o_i \in K_i | (p_{Ks}(o_i) \wedge p_i(o_i)\right) \text{ und } K_i \subset Ks \qquad i,n \in N$$

Wenn es sich um eine strenge Hierarchie handelt, gilt weiterhin

$$K_{ij} \bigcap K_{ik} = \varnothing \text{ mit } k \neq j \qquad i,j,k \in N$$

Derart organisierte Systeme von Klassen verfügen über die Eigenschaft der **Vererbung**.

MATURANA erklärt Vererbung als: „... die transgenerative Invarianz irgendeines strukturellen Aspektes in einer Abstammungslinie miteinander verbundener Einheiten". [MAT87]

Abb. 6-9 Vererbung aus BOOCH ([BOO94])

Es lassen sich zwei Formen der Vererbung unterscheiden, bei denen der Wirkungsmechanismus aber gleich ist:

a) Vererbung Klasse ⟹ Instanz
   Alle Instanzen der Klasse (also des allgemeinen Begriffs) verfügen (erben) über alle Merkmale ihrer Klasse. Eine ganz konkrete Treppe erbt alle Eigenschaften, die in der Begriffsbildung zur Klasse der *Treppen* definiert wurden. Insbesondere gilt dies auch für zukünftige Objekte (zu planende Bauwerksteile).

b) Vererbung Superklasse ⟶ Subklasse

   Alle Unterbegriffe (Subklassen) erben analog zu Pkt.a alle Eigenschaften ihrer Oberbegriffe (Superklassen). Alle Aussagen, die für die Klasse der *Treppen* gültig sind, gelten somit automatisch auch für *Wendeltreppen*.

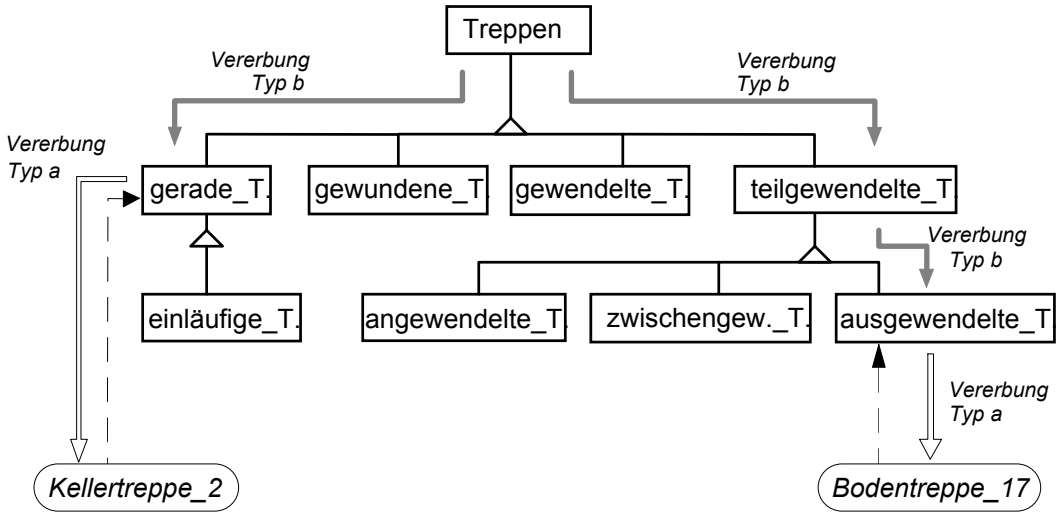
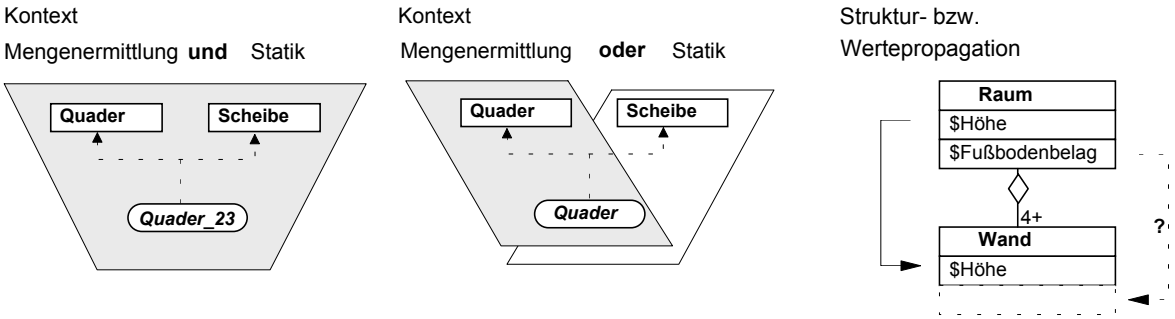


Abb. 6-10 Formen der Erbschaft in Taxonomien

Mit dem Prinzip der Vererbung ist eine effiziente und redundanzarme Form der Wissensorganisation möglich. Wissen wird in der Taxonomie nur einmal dort abgelegt, wo es für die abgeleiteten Sachverhalte Gültigkeit erlangen kann.
Dies ist ein wesentlicher Grund ihrer Nutzung in der OOP. Taxonomien werden dort durch ein hierarchisches (oder auch heterarchisches) Typsystem abgebildet, in dem Strukturierung (Eigenschaften) und Verhalten (Methoden) von Gruppen von Objekten verallgemeinert sind. Genutzt wird lediglich der Rationalisierungseffekt beim Codieren, in dem der Code an möglichst allgemeiner Stelle der Hierarchie angeordnet wird. Um diesen Rationalisierungseffekt noch zu steigern, wird in der OO-Programmierung häufig multiple Vererbung genutzt. Da hier Subklassen von mehreren Superklassen abgeleitet werden können, handelt es sich bei der Taxonomie nicht mehr um einen Baumgraphen, sondern um einen ‘gerichteten azyklischen Graphen’. Hier treten jedoch Problemen der Konfliktlösung auf, wenn Eigenschaften auf verschiedenen Erbschaftspfaden mit unterschiedlichen Eigenschaften versehen werden können. So zeigen Experimente, daß wenn eine mehrfache Zuordnung gegeben scheint, eher eine wechselnde Zugehörigkeit, also ein Kontextswitching vorliegt (Abb. 6-11 a, b).

In einigen OO-Entwicklungssystemen wird der Begriff der Vererbung sogar noch weiter gefaßt. So unterstützt z.B. NEXPERTOBJECT[8] 'Vererbung' entlang von Aggregationsrelationen. In diesem Zusammenhang wäre der Terminus der 'Propagation' eher angebracht, da es sich bei den Teilen, die das Ganze bilden, und dem Ganzem selbst, um völlig verschiedenartige Dinge handelt. Hierbei kann zwar strukturelle Invarianz gegeben sein (Abb. 6-11 c), aber eben keine Relation, die Erbschaft begründen würde.



a.) multiple Vererbung b.) multiple Vererbung als Kontextswitching c.) Propagation entlang einer hierarchischen Relation

Abb. 6-11 Interpretationen von Vererbung

In der Praxis genutzte CAD-Systeme sind heute nur selten objektbasiert und nicht oder nur implizit klassenbasiert (siehe [HEIN93]). Sie sind am ehesten dort anzutreffen, wo eine Konfiguration aus einem festen Set von Basiselementen ausgeführt wird. Doch ist die auf der Generalisierungsrelation beruhende Begriffsstrukturierung in Hierarchien dem frühen architektonischen Entwurf überhaupt angemessen? Während es für Informationssysteme und Thesauri hierzu umfangreiche Untersuchungen gibt (vergl. [ING92], [MEC94], [RAS94]), ist diese Frage für CAD-Systeme wenig untersucht (etwa [GAR94]) und das oft lediglich im Zusammenhang mit Nutzeroberflächen ([GI92]).

Um zu einer empirischen Validierung des Ansatzes zu kommen, wurde ein Experiment entworfen (siehe Anhang B), bei dem durch Probanden 98 Begriffe des Bauwesen so geordnet werden sollten, daß deren inhaltlichen Beziehungen gut wiedergegeben werden. Die Begriffe selbst umfaßten Klassen sehr verschiedener Abstraktionsstufen ('Bauwerk', 'Stuhl'), verschiedener architektonischer Aspekte ('Rezeption', 'Dachsparren'), bis hin zu speziellen Instanzen ('Eiffelturm'). Es nahmen 9 Personen teil (ein Student, 3 Bauing., 5 Architekten), die keine Vorkenntnisse in objektorientierter Analyse hatten. Lediglich 2 Bauing. (B1,B2) und 2 Architekten (A1, A2) beendeten jedoch den Test, da der Aufwand von allen unterschätzt wurde[9] und oft keine Struktur gefunden wurde, die den Probanden selbst befriedigte und von ihm als Endergebnis anerkannt wurde.

Die Auswertung des Experiments bestätigte die Vermutung, daß Superklassen-Subklassen-Instanzbeziehungen eine natürliche Strukturierung begrifflichen Wissens in der Architektur bildet. Auch wenn für eine Verallgemeinerung keine ausreichende Grundgesamtheit gegeben ist, kann diese Wissensorganisation doch als konsensfähig eingeschätzt werden. Alle Probanden wählten eine Generalisierungsbeziehung mit wenigstens zwei Hierarchiestufen als wesentliches Strukturierungsmittel der Begriffe. Begriffe gleichen Allgemeinheitsgrades wurden auf gleicher Stufe der Hierarchie angesiedelt. Die Bauingenieure gingen stärker analysierend vor und wählten eine tiefere Hierarchisierung. Begiffe des funktionalen

[8] NEXPERTOBJECT ist eine KI-Toolkit der NEXUS GmbH
[9] Bemerkung: auch vom Autor

Entwerfens bereiteten ihnen Schwierigkeiten (‘Rezeption’, ‘Kinderzimmer’). Die Architekten wählten entkoppelte, flachere Hierarchien. Von allen wurden neben der Generalisierungsrelation auch die Aggregationsrelation zur Strukturierung genutzt (‘Treppe’ ← ‘Stufe’) sowie weitere Relationen, deren Semantik aber nicht präzise zu fassen war (‘Lagerbereich’, ‘Zufahrt’). Daß es den Probanden offensichtlich Schwierigkeiten bereitete, Relationen mit einer benennbaren Semantik zu versehen oder auch nur die Aggregations- und Generalisationsrelationen sauber zu trennen, fiel bereits in früheren Untersuchungen auf ([EHL93]).
Von allen wurden einfache Hierarchisierungen verwendet. Falls eine mehrfache Zuordnung nötig erschien, wurden hierfür verschiedene Relationentypen verwendet. Die Benutzung verschiedener Relationentypen wurde als Kontextwechsel erklärt. Die enthaltenen Instanzen wurden entweder korrekt klassifiziert oder aber in einem Fall einer eigenständigen Gruppe ‘Beipiele’ zugeordnet.

		Min	*Max*	∅
Zeitaufwand	[h]	1	3	2.25
gewählte Basisklassen	[Anzahl]	1	5	3.25
max. Strukturtiefe (ohne Instanzen)	[Anzahl]	3	5	3.75
hinzugefügte Begriffe	[Anzahl]	0	6	2.75
weggelassene Begriffe	[Anzahl]	7	25	14.75

Tab. 6-2 Ergebnisse des Tests zur Begriffssystematisierung

Die Analyse der Ergebnisse zeigt, daß Klassifizierung und Generalisierung dem Architekten intuitiv vertraut sind. Ihre Semantik wurde von allen Probanten gleich verstanden, was für die von ihnen kreierten assoziativen Relationen nicht gegeben war. Diese Form der Strukturierungen ist somit subjektiv.
Wenn auch die Interpretation der Relationen, welche die Taxonomie konstituieren, offensichtlich gleich ist, so zeigt das Experiment doch auch, daß die Struktur der aufgebauten Begriffshierachien verschieden ist (siehe Strukturierungstiefe). Gerade in den subjektiven frühen Entwurfsphasen kann deshalb die Klassenhierarchie nicht vom CAD-Systementwickler allgemeingültig vorstrukturiert werden. Entsprechende Software sollte es dem Architekten also gestatten, Wissen in das System einzubringen, in der vorgeschlagenen Ordnung zu strukturieren und sich damit sein ’privates’ CAD-System zu schaffen.

6.2.4 Attribute – Beschreibung von Objekten

Attribute beschreiben Eigenschaften und Zustände abstraker Klassen, jedoch auch deren konkreter Ausprägungen (Instanzen). Es werden keine Kenntnisse über weitere Objekte vermittelt, d.h. bei Attributen handelt es sich um reflexive Beschreibungen. Beispiel für Attribute sind Merkmale wie Gewicht, Farbe, Katalognummer und Position.
Eine Stärke des objektorientierten Modellierungsparadigmas liegt in der Kapselung solcher mit einem Objekt assoziierten Informationen. Das Geschlossenheitsmerkmal, das zur Objektfindung lt. MATURANA notwendig ist, zeichnet auch Objekte auf programmiersprachlicher Ebene aus. Aus Sicht dieser Sprachen sind Objekte Aggregationen von Merkmalen und Verhalten (siehe BOOCH). Auf Klassenebene wird ein Attribut lediglich definiert, in dem Name und Typ angegeben werden, während der eigentliche Werteintrag bei konkreten Instanzen

erfolgt. In objektorientierte Wissensrepräsentation wie FRAMES werden Attribute in *Slots* abgelegt. Slots verallgemeinern Attribute, Relationen und Methoden[10] .

Für eine Unterscheidung von Attributen können sie aus semantischer Sicht oder syntaktisch-struktureller Sicht klassifiziert werden.

Die *semantische Klassifikation von Attributen* ist die Sicht des modellierenden Architekten. So liefert etwa das SI-Einheitensystem[11] eine Normierung von Skalaren wie z.B. Länge, Kraft, Druck, Temperatur. In [JOE76], [LAS89], [RIT92] ist eine Grundlage für die Klassifikation von Attributen in der Architektur zu finden (siehe Tabelle 6-3).

Skalenart	*Fixpunkte*	*Definierte Relationen*	*Beschreibung*	*Beispiel*
Nominalskala	alle bzw. keine ausgezeichneten	= ≠	Werte sind diskrete Alternativen, keine Ordnung	Waren-kataloge
Ordinalskala	alle bzw. kein ausgezeichneter	= ≠ >	Werte sind diskrete Alternativen, es existierte eine Rangfolge	Härtegrade
Kardinalskalen				
Intervallskala	Einheiten, Nullpunkt	= ≠ > -	skalare Größe, Nullpunkt und Einheiten werden willkürlich definiert	Temperatur
Differenzskala	Nullpunkt	= ≠ > - +	skalare Größe, Nullpunkt will-kürlich und Einheiten natürlich definiert	Datum
Verhältnisskala	Einheiten	= ≠ > - + : ·	skalare Größe, Nullpunkt natürlich und Einheiten willkürlich definiert, alle arithmetischen Operationen können ausgeführt werden	Länge

Tab. 6-3 Skalen im architektonischen Entwurf (nach [JOE76])

In Tabelle 6-3 werden Skalen für die Beschreibung von „Meßgrößen“ unterschieden ([JOE76]). JOEDICKE verallgemeinert diese jedoch bis hin zu Nominalskalen, bei denen keine Ordnung der beschreibenden Elemente mehr gegeben ist. Laut JOEDICKE dominieren in frühen Entwurfsphasen Wertefixierungen auf Nominalskalen, während später eher auf Kardinalskalen gearbeitet wird.

Die syntaktisch-strukturelle Klassifikation von Attributen unterscheidet sie nach ihrem Datentyp. Hierbei wird keinerlei Semantik eines Attributes vermittelt, sondern deren formaler Aufbau betrachtet. Dieser Aufbau richtet sich im Falle von Programmiersprachen nach den vorhandenen Basisdatentypen (wie character, float, byte etc.) und den Möglichkeiten der Sprache, aus diesen terminalen Basisdatentypen komplexere Typen zusammenzusetzen (string, array, liste, record, set etc.). So lassen sich Merkmale, wie Position oder Steifigkeitsmatrizen, als terminale Attribute von Objekten abbilden. Lediglich SMALLTALK behandelt diese

[10] KappaPC kennt Slots und Methoden, in ALOS (siehe Anhang D) ist Slot Oberbegriff zu Methode und Attribut.
[11] Système International d'Unités

beschreibenden Elemente wiederum als eigenständige Objektklassen und ordnet sie folgerichtig wiederum in Hierarchien. (siehe Abb. 6-12)

Im Rahmen von CAD-Systemen, aber auch von Datenbanken werden dem Nutzer i. allg. nur die syntaktisch-strukturellen Attributetypen angeboten. Dies hat zur Folge, daß der Nutzer bei der Beschreibung von Artefakten die Sicht des Programmierers einnehmen muß. Will er ein Attribut 'Querschnitt' bei einer Objektklasse 'Balken' definieren, so muß er dieses Attribut als vom (Daten-)Typ 'float' definieren, während die natürliche Beschreibung doch vom Typ 'Fläche' wäre. Die Verfügbarkeit derartiger semantischer Attributtypen würde es auch ermöglichen, weitere Informationen an sie zu binden, wie z.B. Maßeinheiten. Hiermit wäre ein eleganter Weg gegeben, durch Erweiterungen dieser Typhierarchie semantische Attribute als Extensionen strukturell-syntaktischer darzustellen.

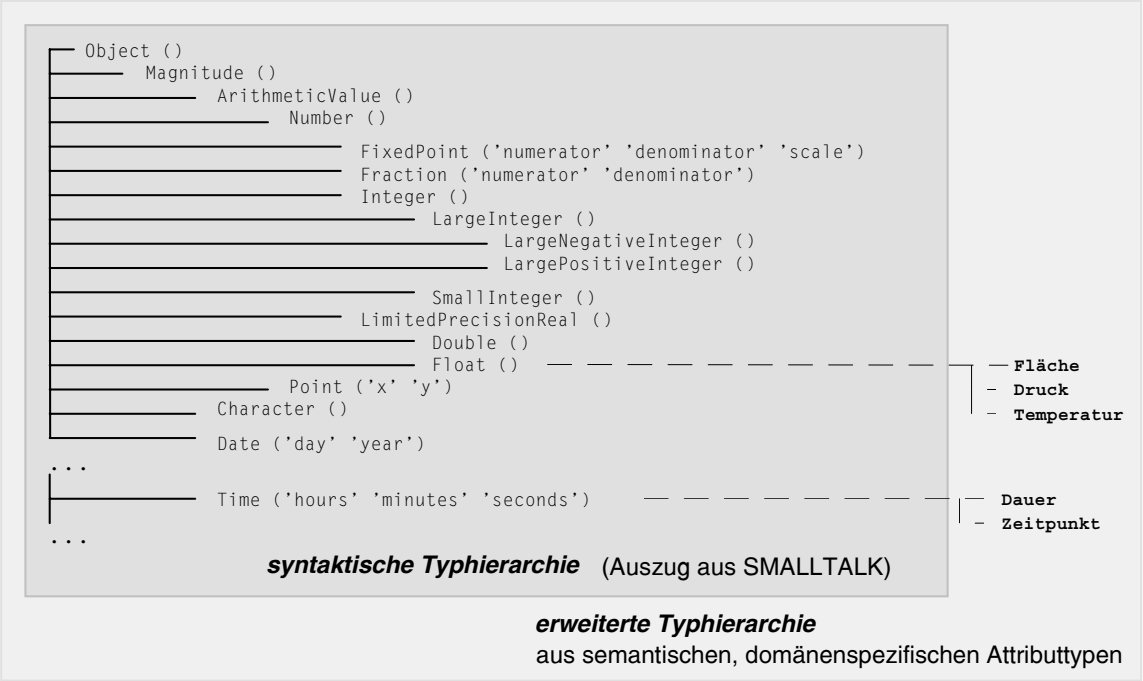


Abb 6-12 Domänenspezifische Attributtyphierarchien

6.2.5 Relationen – Beziehungen zwischen Objekten

Relationen[12] sind Beschreibungen von 'Interobjektzusammenhängen'. Sie sind referenzierende Merkmale von Objekten und als solche immer mit Objekten assoziiert, d.h. sie können nicht losgelöst von diesen existieren, umgekehrt jedoch sehr wohl.
Wie bei Attributen wird eine Relation auf Klassenebene lediglich definiert, indem Name, Art der Relation und ggf. der referenzierbare Objekttyp (bei OOP) angegeben werden. Die Instanziierung der Relation, d.h. deren eigentliche Applikation, erfolgt zwischen konkreten Objektinstanzen (Abb. 6-13).
Während die Objektorientierung die Objekte in den Mittelpunkt der Strukturierung von Modellwelten stellt, sind im Relationenkonzept[13] die Zusammenhänge zwischen den Objekten ein wesentliches Strukturierungsmittel.

[12] der verwendete Relationenbegriff entspricht nicht der Begriffsverwendung bei relationalen Datenmodellen
[13] eine umfassende Darstellung ist etwa in [SAU92] zu finden.

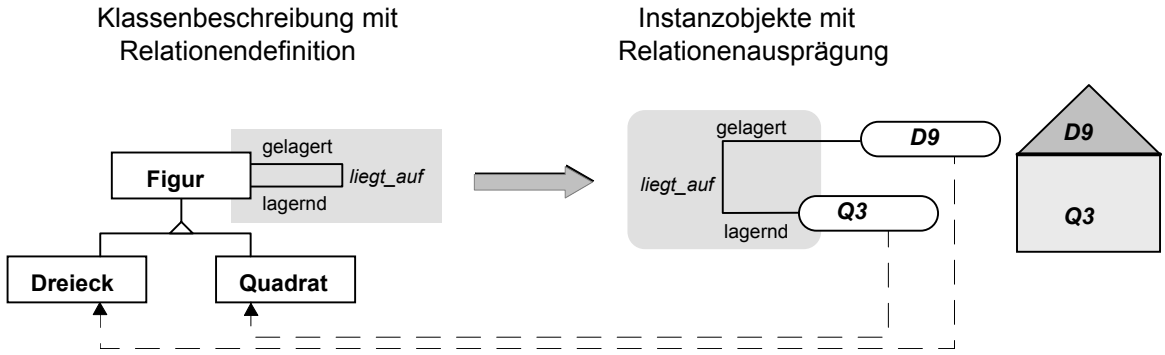


Abb. 6-13 Abstrakte und beobachtbare Relationen

In Form von Koppeltabellen werden dort Relationen eines Typs getrennt von den ebenfalls in Tabellen organisierten Objektbeschreibungen geführt. Hieraus ergibt sich der 'independence mismatch', da beide Konzepte schwer in einer (Programmier-) Sprache zu vereinen sind (vergleiche [HEU92]). So fand das Relationenkonzept mit wenigen Ausnahmen (MODULA-R) keinen Eingang in die Programmiersprachen und dient heute als Basismodell für relationale Datenbanken.

Für komplexe CAD-Modelle ist die objektzentrierte Modellierung ausdrucksstärker und stellt damit das bessere Modellierungsparadigma dar. HÄRDER ([HÄR89]) führt hierzu aus:

> „Klassische Datenmodelle (wie das relationale, Anm. des Autors) erheben zwar den Anspruch, allgemeingültig zu sein – sie sind es aber bestenfalls für die Modellierung administrativ-betriebswirtschaftlicher Anwendungen, die bei ihrem Entwurf Pate standen."

Für die verschiedenen Anwendungsfelder der Informationstechnologie läßt sich ein verschiedener Grad der Modellstrukturierung mittels Relationen konstatieren. Die meisten Programme wurden zur Verwaltung großer Objektmengen geschaffen, besonders im kaufmännischen Bereich oder für Informationssysteme. Sie bieten nur wenig Relationentypen, die aber sehr häufig im konkreten Modell eingesetzt werden. Für CAD-Anwendung ist jedoch weniger die Menge der Ausprägungen einer Relation, sondern eher die Anzahl der Relationenarten selbst von Bedeutung. Sie ist hier vergleichsweise hoch. Hieraus ergibt sich die in Abbildung 6-14 hervorgehobene Entwicklungstendenz zukünftiger CAD-Systeme.

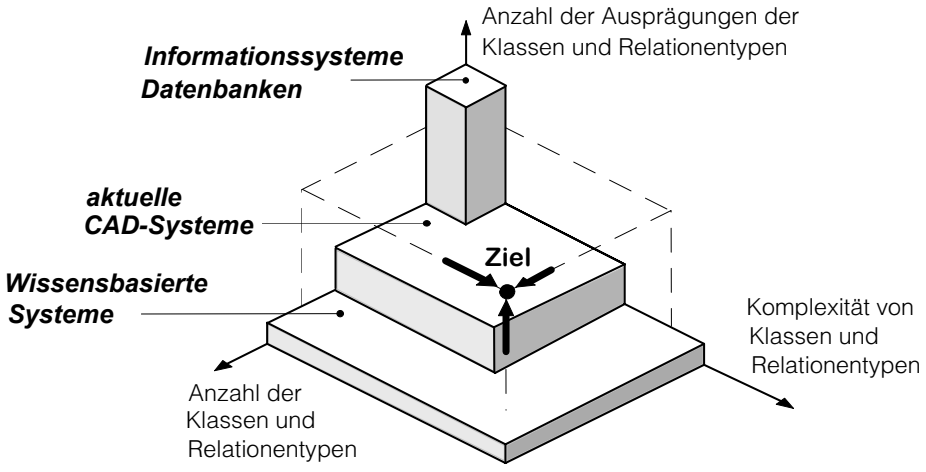


Abb. 6-14 Ziel der Entwicklung dynamischer CAD-Systeme

Für den Architekten als Nutzer von CAD-Systemen stehen zum modellierenden Entwerfen heute lediglich die Relationenarten zur Verfügung, die der Programmierer in der Software vorgedacht und fixiert hat. Um in einer gestalterischen Umgebung dynamisch neue Relationsarten erzeugen zu können, ist deren Klassifikation sinnvoll. Analog zur Klassifikation von Attributen ist eine Unterscheidung nach *strukturell-syntaktischen* und *semantischen* Gesichtspunkten möglich.

SYNTAKTISCH-STRUKTURELLE KLASSIFIKATION VON RELATIONEN

Die Betrachtung des strukturell-syntaktischen Aspektes betont vor allem die technische Umsetzung in Datenbanken und Programmiersprachen. Die Relationsarten werden hier nach ihrem Aufbau unterschieden. Dies betrifft die Anzahl der in Beziehung gesetzten Objekte, ob die Relation eine Richtung besitzt, ob sich die Rollen von Objekten in der Relation unterscheiden lassen, ob die Relation bewertet ist und wenn ja wie, sowie weitere Informationen, die mit der Relation assoziiert sein können. Auf dieser formalen Ebene lassen sich auch Bedingungen notieren, die die Struktur der entstehenden Relationengraphen betreffen, wie etwa der Ausschluß zirkulärer Relationen.
In OO-Programmiersprachen und Modelliermethodiken steht häufig wieder der prozessorale Aspekt im Vordergrund. So werden in [BOO94] die 'Rollen', welche die Objekte in gerichteten Relationen einnehmen, als „Denotation einer Auswahl von Verhalten (Methoden, Anm.d.A.), die zu einem einzelnen Zeitpunkt wohl definiert sind." beschrieben. In der OOP mittels getypter Programmiersprachen wie C++ werden Relationen als getypte Pointer in Membervariablen realisiert. Zum Zeitpunkt der Typdeklaration (also der Klassenbildung) muß der Typ von Objekten, auf die verwiesen werden kann, fixiert werden. Dies unterstreicht noch einmal die verschiedene Semantik von Relationen im Rahmen des modellierenden Entwurfs und der generalisierenden Modellierung einer Domäne.

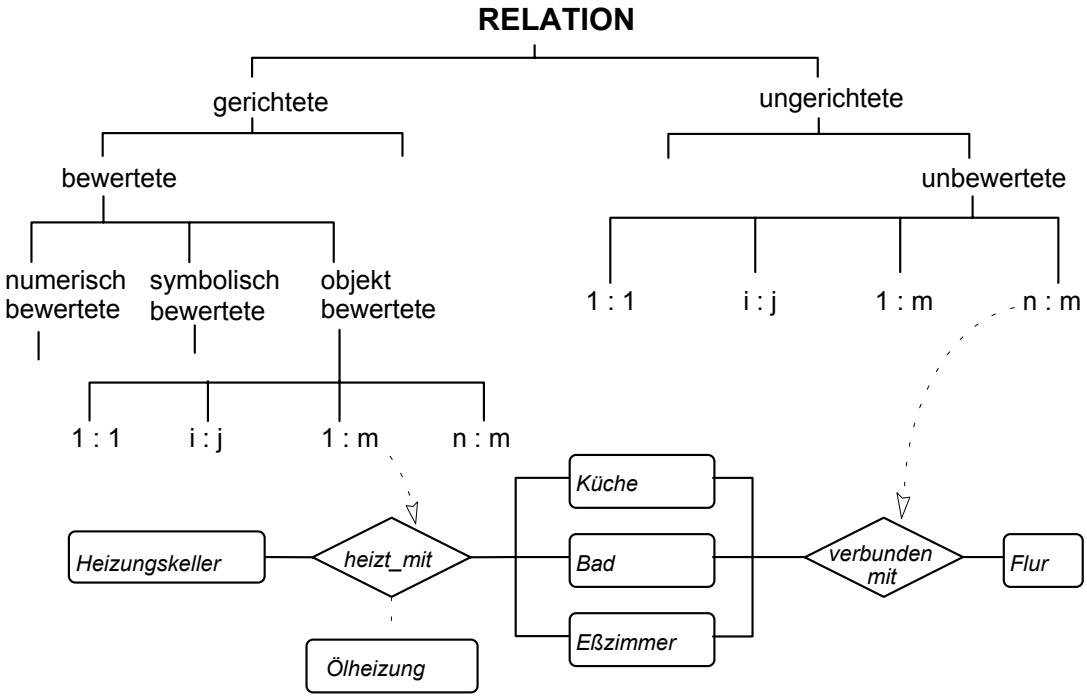


Abb. 6-15 Klassifikation von strukturell-syntaktischen Relationentypen

SEMANTISCHE KLASSIFIKATION VON RELATIONEN

Ausgangspunkt der semantischen Klassifikation von Relationenarten sind die paradigmatischen Relationen der jeweiligen Modellierungsmethodik. So benennt SAGER ([SAG91]) im Kontext objektbasierter, ingenieurtechnischer Modellierung als paradigmatische Relationstypen die Generalisierung, die Instanziierung sowie eine Aggregationsrelation, die dem natürlichem Aufbau eines Ganzen aus Teilen entspricht. VIEGENER ([VIE93]) definiert im Bereich der Informationssysteme ganz ähnlich eine Abstraktionsrelation (Subklassenbildung) sowie eine Bestandsrelation in dem Sinne, daß ein abstrakter Thesauruseintrag eine Menge von Dokumenten (Instanzen, siehe Tab. 6-4) repräsentiert.

CAD-Modellen ([SAG91])	**Thesauri von Informationssystemen** ([VIE93], DIN1463)
I. *Generalisieren/Spezialisieren* Dieser Beziehungstyp gestattet den Aufbau von Objekthierarchien, er gilt zwischen Objektklassen und beschreibt das Verhältnis Super-/Subklasse.	*Abstraktionsrelationen* Stellt die klassische Hierarchierelation dar und beschreibt Verhältnis Ober-/ Unterbegriff.
II. *Klassifikation/Instanziierung* Über diesen Beziehungstyp werden konkrete Ausprägungen von Objektklassen erzeugt (Instanziierung). Er beschreibt das Verhältnis Klasse - Ausprägung.	*Bestandsrelationen* Sie stellt die klassische Hierarchierelation in Thesauri dar und beschreibt die Relation zwischen dem Ganzen (hier allerdings ein abstrakter Begriff!, Anm. d. A.) und seinen Bestandteilen (hier den Dokumenten)
III. *Aggregation/Zerlegung* Als Aggregation bezeichnet man das Zusammenfassen von Einzelobjekten zu einem übergeordneten, komlexer aufgebauten Objekt. Die Aufbaustruktur solcher komplexer Objekte wird auch als Kompositionshierarchie bezeichnet. Dieser Beziehungstyp gilt zwischen Instanzen.	
IV. *Assoziation* Als Assoziation werden alle Beziehungen zwischen voneinander unabhänigen Objekten bezeichnet, die nicht einem der o.g. Typen zugeordnet werden können.	*Assoziationsrelation* Repräsentiert die Verbindung zwischen Begriffen, die weder äquivalent noch eindeutig hierarchisch verbunden sind
	Äquivalenzrelationen Beziehung zwischen gleichwertigen Begriffen (Synonymie)

Tab. 6-4 Semantische Relationstypen in CAD-Modellen und Thesauri

Die aus Informationssystemen bekannte Äquivalenzrelation ist im Rahmen einer CAD-Modellierung unbekannt, aber vorstellbar und leicht realisierbar, solange es sich um reine Synonymie handelt. Dies könnte mnemonische Probleme in der Gruppenarbeit lösen helfen. Eine multiple Zuordnung entlang der Spezialisierungs- bzw. Instanziierungsrelation vermag dies nicht zu leisten.

Von besonderer Bedeutung für CAD-Systeme ist die Relation '*has_part*' bzw. invers '*part_of*' (III). In technischen Anwendungsgebieten sollten von vornherein alle dem Nutzer verfügbaren Klassen über die Aggregationsrelation verfügen, da sich technische Produkte einiger Komplexität immer rekursiv in Teilobjekte zergliedern lassen. Die entstehende Aggregationsstruktur kann als heterarchischer Graph dargestellt werden[14]. Alle Knoten dieser Struktur sind Instanzen. Im Entwurfsprozeß wird der Wurzelbereich dieser Modellstruktur bereits in der Konzeptphase erstellt. Der Wurzelbereich hat Bestand bis zum Abriß des Gebäudes und läßt sich über alle Phasen des Entwurfs vom Konzipieren und groben Gestalten, über die Detailkonstruktion bis zur computergestützten Bewirtschaftung und zur Organisation des Datenaustausches nutzen. Konzipieren besteht wesentlich aus dem Fixieren dieser Struktur.

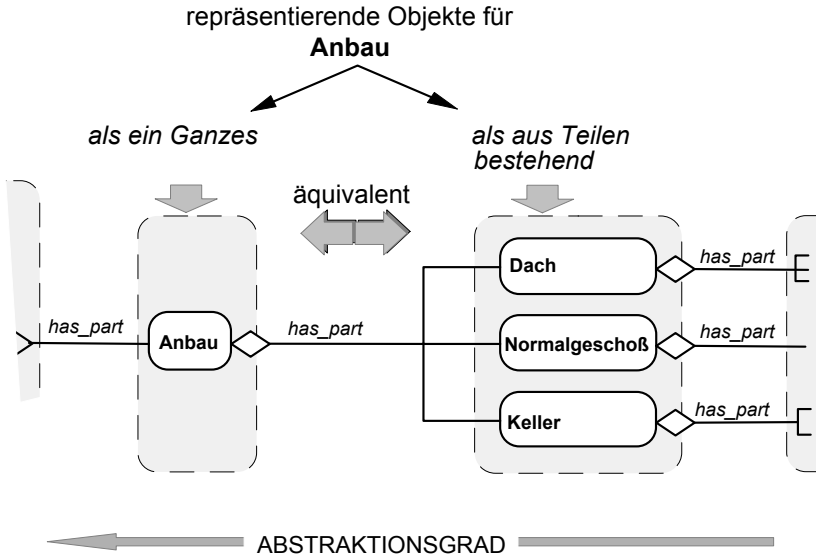


Abb. 6-16 Abstraktion in der Kompositionshierarchie

Die Aufbaustruktur gestattet es, bei Bedarf ein Objekt als Ganzes zu betrachten oder als aus Teilen aufgebaut und so den Abstraktionsgrad des betrachteten Entwurfsproblemes zu variieren. Sie widerspiegelt die Art der Problemdekomposition, um das Komplexitätsniveau bei der Problemlösung zu senken und hat folglich große Bedeutung für Modellorganisation, Modellaustausch und für die Organisation des Entwurfsprozesses selbst.
Trotz ihrer Bedeutung wird eine explizite Aggregationsrelation weder in OO-Programiersprachen noch in generischen CAD-Systemen angeboten. Lediglich Systeme zur Erstellung wissensbasierter CAD-Lösungen verfügen über sie, wie etwa PLAKON ([CUN91]). Sie sind

[14] heterarchisch heißt ein azyklischer, hierarchischer Graph mit gerichteten Kanten, wobei für einen Knoten mehrere Vorgängerknoten existieren können.

dort Basis von Verfahren zur Skelettkonstruktion[15]. CAD-Systeme bieten heute lediglich die Möglichkeit, rekursiv Mengen von Konstruktionselementen zusammenzufassen. Das vordergründig Ziel ist es hier, geometrische Operationen auf eine Menge von Objekten anzuwenden oder weitere Informationen mit solchen Gruppen zu assoziieren. Die Gruppen selbst sind aber keine echten Modellobjekte, wie beispielsweise AutoCAD und einige seiner Aufsatzapplikationen zeigen. Solche hierarchischen Mengen von Modellelementen können nur 'bottom-up' erzeugt werden.

Doch auch Nutzer haben mit der 'part_of'-Relation Schwierigkeiten. Sie wird häufig zur Darstellung jeglicher hierarchischer Beziehung zwischen Modellelementen benutzt, die eigentliche Semantik der Relation wird nicht repräsentiert. Ein Beispiel hierfür ist der Aufbau eines Raumes aus Wänden, der häufig als klassisches Beispiel einer Aggregationsstruktur angegeben wird. Hier wäre es angebracht, von einer 'bildet'-Relation zu sprechen, da Wände einen Raum eben bilden und nicht eigentlich Teil von ihm sind. Ein weiteres Problem stellt die lt. Definition vorhandene mehrfache Zuordenbarkeit eines Teils zu Komplexobjekten dar. Dies hat zur Folge, daß kummulative oder geometrische Operationen auf den Komplexen nicht ohne weiteres korrekt arbeiten. (siehe Abb 6-17)

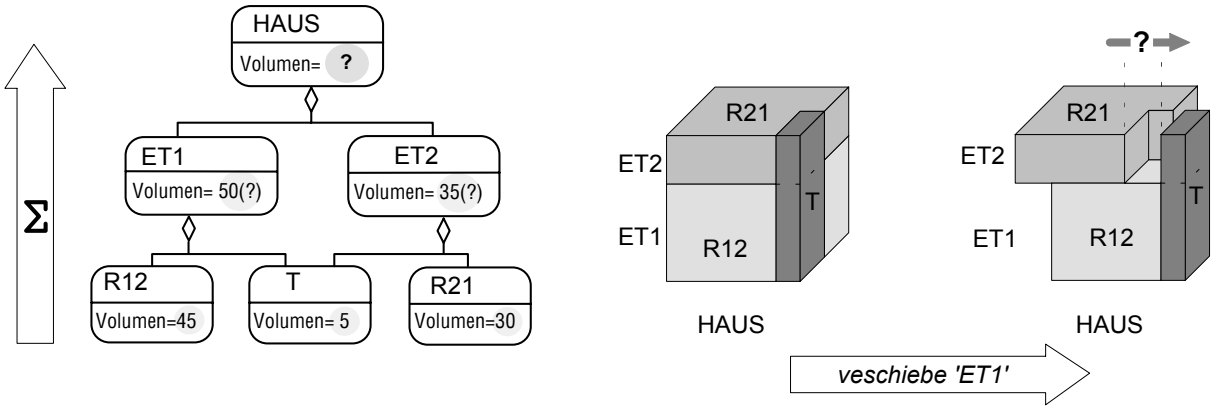


Abb. 6-17 Kummulative und geometrische Operationen in Aggregationen

RELATIONEN IN FRÜHEN ENTWURFSPHASEN

Eine semantische Analyse von Relationen, die in Modellen des frühen Entwurfs relevant sind, gelingt nur bedingt. Ziel wäre es, eine Basisklassifikation entsprechend der Entwurfsaspekte funktionaler, konstruktiver und formorientierten Entwurf zu erstellen. Abgesehen von formal begründbaren toplogischen oder geometrischen Relationen, gibt es jedoch für den funktionalen und formorientierten Entwurf keine anerkannte Klassifikation und damit einheitliche Interpretation von Beziehungen. So führt etwa JOEDICKE ([JOE76]) aus, daß „optimale Raum*beziehungen*" zu spezifizieren seien, gibt aber keine derartigen Beziehungen an, die allgemeingültig wären. Dies liegt wiederum in der Besonderheit des Bauwerksentwurfs begründet, indem wechselnde Nutzung des Bauwerks wechselnde funktionale Beziehungen bedingen.

[15] Skelettkonstruktion ist eine Problemlösemethode für Konstruktionsaufgaben, bei dem Knoten (Teilprobleme) eines hierarchischen Skelettplans expandiert und rekursiv verfeinert werden ([PUP91]).

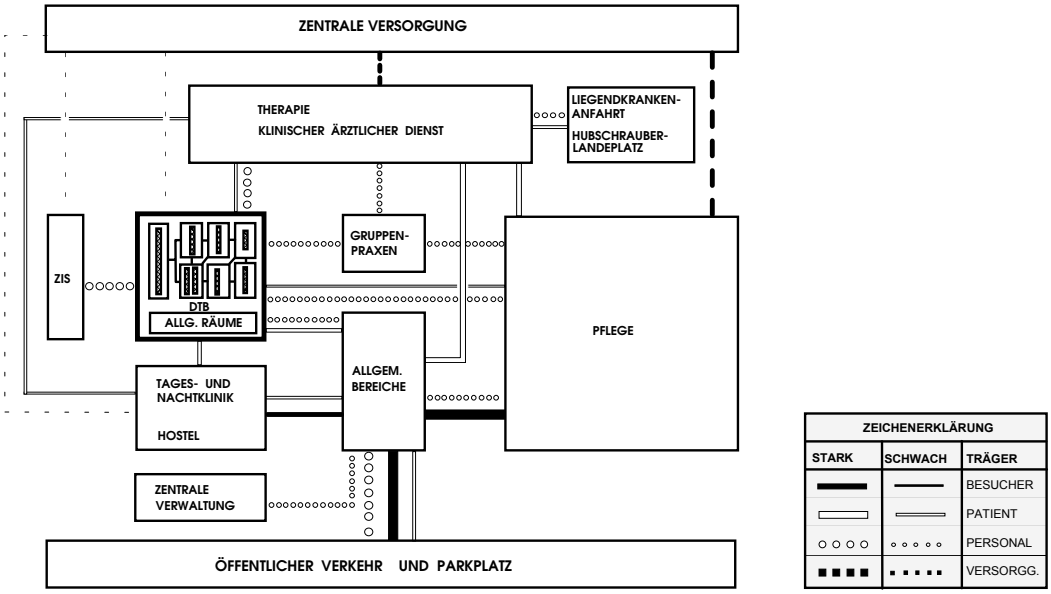


Abb.6-18 Blockdiagramm eines 600-Betten Krankenhauses ([JOE76])

Für den konstruktiv-statischen Entwurf gelingt es eher, ein fixes Set von Relationen zwischen Konstruktionselementen a priori zu bestimmen. Eine häufige Interpretation für die im Rahmen der funktionalen Analyse gefundenen Beziehungen zwischen architektonischen Modellelementen ist, daß mit ihnen die Gültigkeit bestimmter räumlicher Relationen im gestaltorientierten Entwurf gefordert wird. Dies sind z.B. häufig Wege, die Dinge oder Personen bei der Benutzung des Gebäudes zurücklegen oder allgemeine Nachbarschaften von Nutzungseinheiten, die durch die geplante Nutzung bedingt sind. JOEDICKE unterscheidet diese Beziehungen nach Träger der Relation, Intensität, Richtung sowie „weitere substantielle Eigenschaften“ (Abb.6-18).

Eine Computerstützung des konzeptuellen Entwurfs muß es dem Architekten ermöglichen, sein, durch eine funktionale Analyse gefundenen, Entwurfsintentionen zu repräsentieren. Es ist schwer vorstellbar, ihm lediglich strukturell-syntaktische Relationen als Ausdrucksmittel anzubieten, wie dies für den Softwareentwurf gilt. Der Vorrat an Relationentypen sollte dynamisch erweiterbar sein.

Grundlage bilden die beiden Varianten der Klassifikation von Relationentypen. Während die Taxonomie der strukturell-syntaktischen Relationen durch die Möglichkeiten der Programmierwerkzeuge determiniert und damit statisch ist, können semantische Taxonomien technisch einfach dynamisch erweiterbar gestaltet werden. So schlägt etwa KUHLEN [KUH91] eine Linktaxonomie für Hypertext-/Hypermediasysteme[16] vor. Sie dienen in diesem Bereich der Unterstützung von Argumentationsprozessen. In ähnlicher Weise können sie im CAD-Bereich eine Modellinterpretation etwa beim Modellaustausch erleichtern.
Den Kern der semantischen Relationentaxonomie bilden in Abhängigkeit von der zu unterstützenden Anwendungsdomäne formale Relationen (wie etwa geometrische) bzw. konsensfähige semantische Relationen der Domäne. Diese Relationen sind Basis der Softwareentwicklung. Darüber hinaus bilden ‘private’ (d.h. nutzerspezifische) Relationen dynamische Erweiterungen dieses Kernbereichs. Hiermit wird es dem Architekten als Systemnutzer möglich, neues begriffliches Wissen auch auf Ebene der Relationen einzubringen (Abb. 6-19).

[16] über Linktaxonomien verfügt z.B. SEPIA des IPSI-Institutes der GMD ([HAA95]), eine „Typologie von Verknüpfungen“ in Hypertextsystemen schlägt KUHLEN vor ([KUH91]).

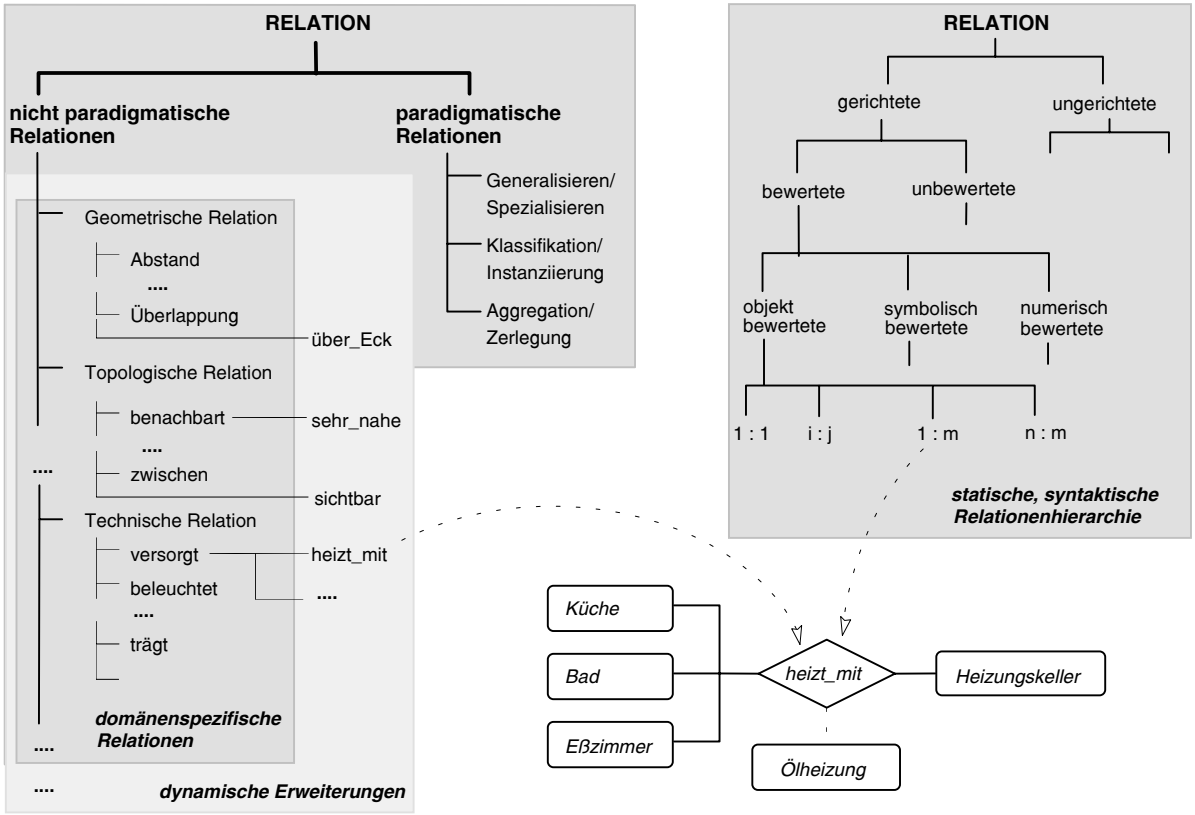


Abb. 6-19 Ableitung domänenspezifischer Relationen

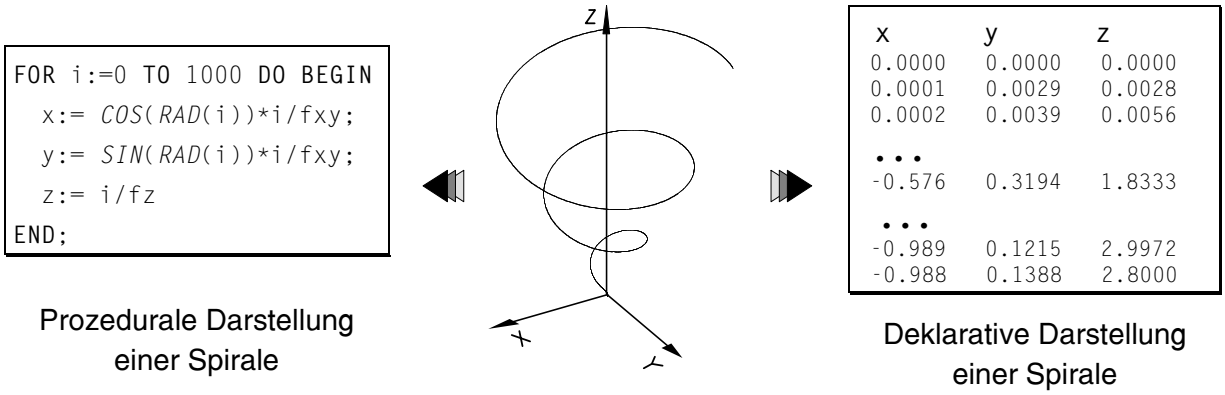
6.2.6 Methoden – Verhalten von Objekten

Methoden sind kein eigentlicher Bestandteil deklarativer Bauwerksmodelle, sondern deren prozedurale, anweisungsorientierte Ausprägung. Ihre Verwendung heißt letztlich immer das Organisieren von Prozessen. BOOCH ([BOO94]) unterscheidet *Methoden* (Methods) und *Verhalten* (Behavior):

- *Methode* – Eine Funktion (Operation), die Teil einer Klassendefinition ist. Eine *Nachricht* (Message, Botschaft) an ein Objekt ist der Aufruf dieser Funktion bei einem Objekt. Im Rahmen der OOP sind Methodentexte nicht zur Laufzeit zugreifbar oder zumindest nicht für einen Zugriff vorgesehen. Sie sind im allgemeinen nicht änderbar und stehen also dem Anwender zur Beschreibung von Artefakten nicht zur Verfügung. Sie können wie Attribute vererbt und lokal in Klassen neu belegt werden (Polymorphismus). Instanzen verfügen jedoch nicht selbst über eigene Methoden, d.h. jede Ausprägung der Klasse verfügt über denselben Satz von Methoden. Die Instanzen sind folglich in einem konkreten Kontext nicht über ihre Methoden differenzierbar.
- *Verhalten* – Die Art und Weise wie ein Objekt in einer Umgebung agiert und reagiert. Dies wird durch die Gesamtheit seiner Methoden, aber auch durch den Modellzustand bestimmt.

Grundsätzlich besteht eine Äquivalenz zwischen einem Modell und der Aktionensequenz, die es erzeugt hat. Obwohl man davon ausgeht, daß deskriptive Modelle ausdrucksstärker,

besser interpretierbar und mit geringerem Speicherbedarf bewahrt werden können, ist ggf. auch die Verwendung einer generierenden Beschreibung sinnvoll. (siehe Abb. 6-20).



Prozedurale Darstellung einer Spirale

Deklarative Darstellung einer Spirale

Abb. 6-20 Äquivalenz der prozeduralen und deklarativen Modellierung

Endpunkt solcher Überlegungen sind Objekte als eigenständige, aktive, intelligente Individuen, die selbständig in einer Modellwelt agieren. Hier steht die Simulation und damit die Prognose des Verhaltens eines Modells im Vordergrund. In [HOV94] wird als weitreichendes Beispiel hierfür das ACTOR-Modell (AGHA 1986) aufgeführt, in dem sich „aktive Objekte (mit Wissen ausgestattete Container) aus eigenem Antrieb sozial organisieren".
Eine Verwendung von Methoden als Beschreibungsmittel im eigentlichen Entwurfsprozeß birgt jedoch Probleme. So kann die Anwendung von Methoden weitreichende Modellveränderungen triggern, deren Komplexität und Konsequenzen durch den Nutzer nur selten beherrscht werden. Um Methoden als Ausdrucksmittel zur Modellierung konkreter Bauwerke anzuwenden, muß der Nutzer sie in einer komplexen, letztlich programmiersprachlichen Syntax formulieren. Es gibt keine Mittel, um die Semantik derartiger Konstrukte zu formulieren und damit auch keine Möglichkeit, deren Korrektheit formal zu sichern. Dies kann zu Fehlfunktionen und Absturz der Programme führen, die die Methoden in einer konkreten EDV-Umgebung ausführen. Diese Programme wären im übrigen Teil des Modells! Eine Unabhängigkeit der Modelle von Betriebssystem- und Rechnerversion ist damit nur schwer zu gewährleisten. Ob Ansätze wie die JAVA-Technologie[17] hier Verbesserungen bringen werden, bleibt abzuwarten. Die ständigen Erweiterungen um proprietäre Elemente wie ACTIVE-X[18] stimmen eher pessimistisch.

Der zur Anwendung von Methoden benötigte Kenntnisstand in der Informationstechnologie ist bei Architekten und Bauingenieuren gegenwärtig (noch) nicht vorhanden. Betrachtet man deren vergleichsweise geringen Anteil an der universitären Ausbildung, so ist er auch in näherer Zukunft nicht zu erwarten. Programmierendes Arbeiten gelingt der Mehrzahl der Nutzer lediglich auf sehr einfach strukturierten Modellen, wie sie etwa in Form der Tabellen bei Tools zur Tabellenkalkulation zu finden sind. Zur Unterstützung des frühen architektonischen Entwurfs sollte deshalb auf die Verwendung von modellverändernden oder modellgenerierenden Methoden als Beschreibungsmittel durch den Nutzer weitgehend ver-

[17] JAVA ist eine betriebssystem- und maschinenunabhängige OO-Programmiersprache von SUN-Soft zum Erstellen von Programmen, die im Internet austauschbar sind (Applets).
[18] ACTIVE-X ist das aktuelle MicroSoft 'Component'-konzept für 'Distributed Object Computing' (DOC). Es dient auch zur Verteilung von Programm bzw. Programmteilen im Internet.

zichtet werden. Durch den Nutzer anwendbar sind prozedurale Modellerweiterungen, die in ihrer Komplexität beschränkt sind. Hierzu zählen einfache Constraints zur Wertepropagation sowie prozedurale Zusätze zu Attributen ('procedural attachments'), um einfache formale Auswertungen wiederum zur Wertepropagation zu ermöglichen. Deren Semantik sollte eingeschränkt und intuitiv einsichtig sein. Geeignet sind hierzu kummulierende Auswertungen über die standardisierte 'has_part'-Relation sowie die Propagation von Gleichheitsconstraints (siehe Abb. 6-21).

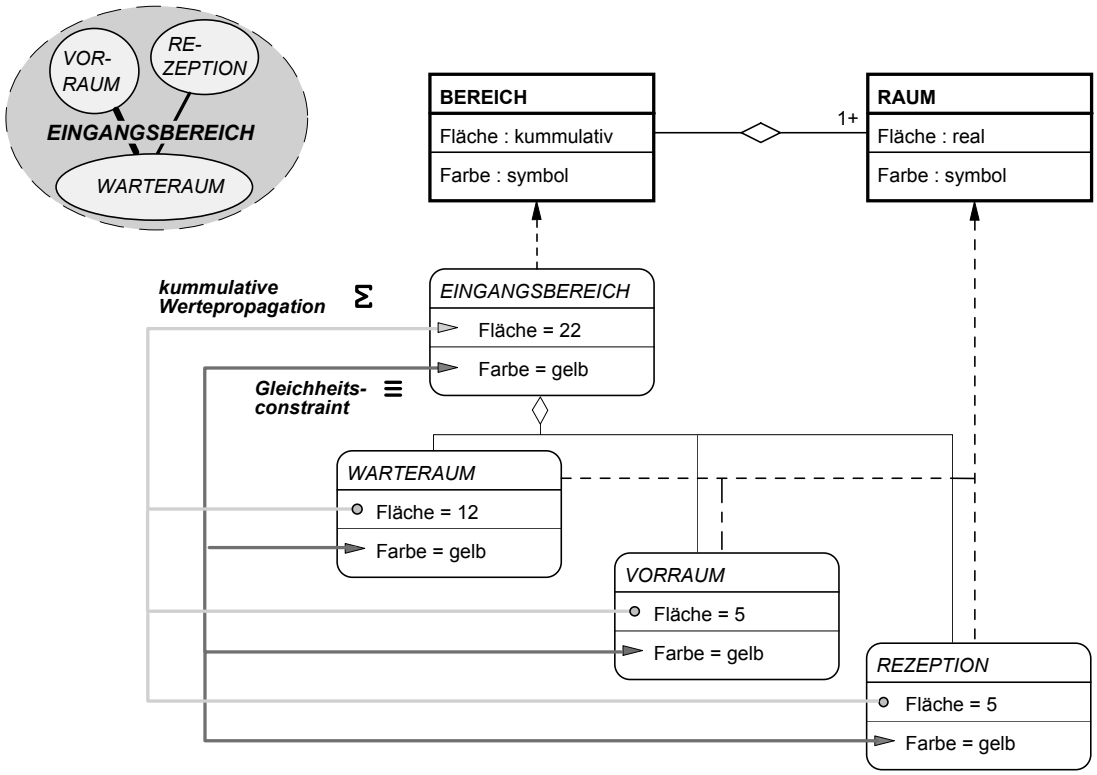


Abb. 6-21 Objektbeschreibung mittels einfacher Constraints und kummulativer Propagation

6.2.7 Facetten – terminale Beschreibungselemente

Eine Modellwelt wird wesentlich durch die Art ihrer terminalen, nicht weiter untersetzbaren Informationen gekennzeichnet. Das Objekt als modellhaftes Analogon zu beobachtbaren oder imaginierbaren Entitäten stellt kein solches Beschreibungselement dar. Es ist eine Kapsel zur geschlossenen Verwahrung aller mit diesem Objekt assoziierten Informationen. In der gebräuchlichen softwaretechnologischen Interpretation der Objektorientiertheit sind diese Informationen Attribute und Relationen, sie werden hier als Terminale des Paradigmas verstanden[19]. Dies ergibt sich letztlich aus der evolutionären Enwicklung objektorientierter Programmiersprachen, deren Typkonzept Klassen als recordähnlichen Verbund verstehen, dessen Felder Attribute und Relationen abbilden.
Für die ingenieurtechnische Beschreibung von Objekten ist dies jedoch unbefriedigend, da dies keine Untersetzung von Attributinformationen zuläßt. Aussagen wie: „Die *Fläche*

[19] auch bei der Verwendung des relationalen (Datenbank-) Modells für CAD-Produktmodelle stellt dies die terminale Stufe von Informationen dar, wie etwa im Molekül-Atom-Modell nach HÄRDER/ MITSCHANG [MIT88], siehe auch RANGLACK [RAN92]

(Attribut) von *Bad* (Objekt) ist *25* (Wert)“ sind aber nur interpretierbar, wenn die Maßeinheit von Fläche im jeweiligen Kontext implizit ist und sie also a priori definiert wurde. Daraus folgt, daß vom Nutzer eines CAD-Systems eine maßeinheitentreue Eingabe erwartet wird. Er kann nicht (oder zumindest nicht im eigentlichen Modell) einem Attribut ein Tupel aus Maßeinheit und Maßzahl zuordnen. Für viele CAD-Systeme stellt dies eine untragbare Einschränkung dar[20] . Auch die beschriebenen Formen attributiver und relationaler Unschärfe erfordern eine weitere Untersetzbarkeit derartiger Informationen.
Die Wissensrepräsentation der FRAMES [MIN75] bietet eine solche Untersetzbarkeit von *Slots* (Vereinheitlichung von Attributen und Relationen) in Form der *Facets* an. Facetten bilden die Terminale dieser Wissensrepräsentation (siehe Abb. 6-22). Die Möglichkeiten zur semantischen Anreicherung von Attribut- und Relationeninformationen sind vielfältig. Hier einige oft verwendete deskriptive Facetten:

- *Wert* - eigentlicher Werteintragung, entspricht dem aus der konventionellen Objektorientierung bekannten Dateneintrag (bei Klassen wäre dies die Standardannahme für die Einträge bei Instanzen der Klasse)
- *Wertevorrat* - besonders bei symbolischen Attributen bzw. Relationen
- *Werteeinschränkungen* - bei numerischen Attributen bzw. Relationen
- *Unschärfebeschreibungen* - verschiedene Facetten zur Beschreibung von Unschärfe, aber auch jeder anderen Art von Vagheit bei Slots. Die benötigten Facetten sind abhängig von Art der Formalisierung der Unschärfe und dem Typ des Slots
- *Maßeinheiten* - bei numerischen Attributen sinnvoll
- *Kommentare* - beliebig mediale Annotationen (i. allg. Strings)
- *Eigentümer* - Eigentümer oder Erzeugender einer Information z.B. als Basis eines Signatur- oder Locking-Mechanismus
- *Kontextabhängigkeiten* - hier lassen sich Wert- oder Wertetypänderungen beschreiben, die bei Kontextwechsel auftreten.

Über diese deskriptive Anreicherung von Slotinformationen hinaus sind sogenannte ‘procedural attachments’ ([BUN90]) geeignet, die Funktionalität entsprechender Programmiertools zu erweitern. So verfügen etwa die Systeme KappaPC und NexpertObject über sogenannte Monitore, die als Facetten implementiert sind und die als Trigger lesenden und schreibenden Zugriff auf Slotinformationen überwachen [KAP20]. In ähnlicher Weise lassen sich etwa Constraints[21] implementieren [BAB30]. Die Struktur von Slotobjekten, d.h. deren Facettisierung, ist allerdings auch in den gennaten Systemen fixiert. Die Facetten der verfügbaren Attribut- und Relationentypen können weder durch den Anwendungsprogrammierer noch durch Nutzer verändert oder erweitert werden. Obwohl das für den Programmierer eine Einschränkung darstellt, ist es für den Nutzer tolerabel.

[20] etwa das System XKL [KOP93] zur Layoutplanung von Airbuskabinen, die in Deutschland in ‘mm’, international aber in ‘inch’ konstruiert werden.
[21] Zwangsbedingungen, die den möglichen Wertvorrat einschränken oder Werte in Abhängigkeit anderer Werte determinieren

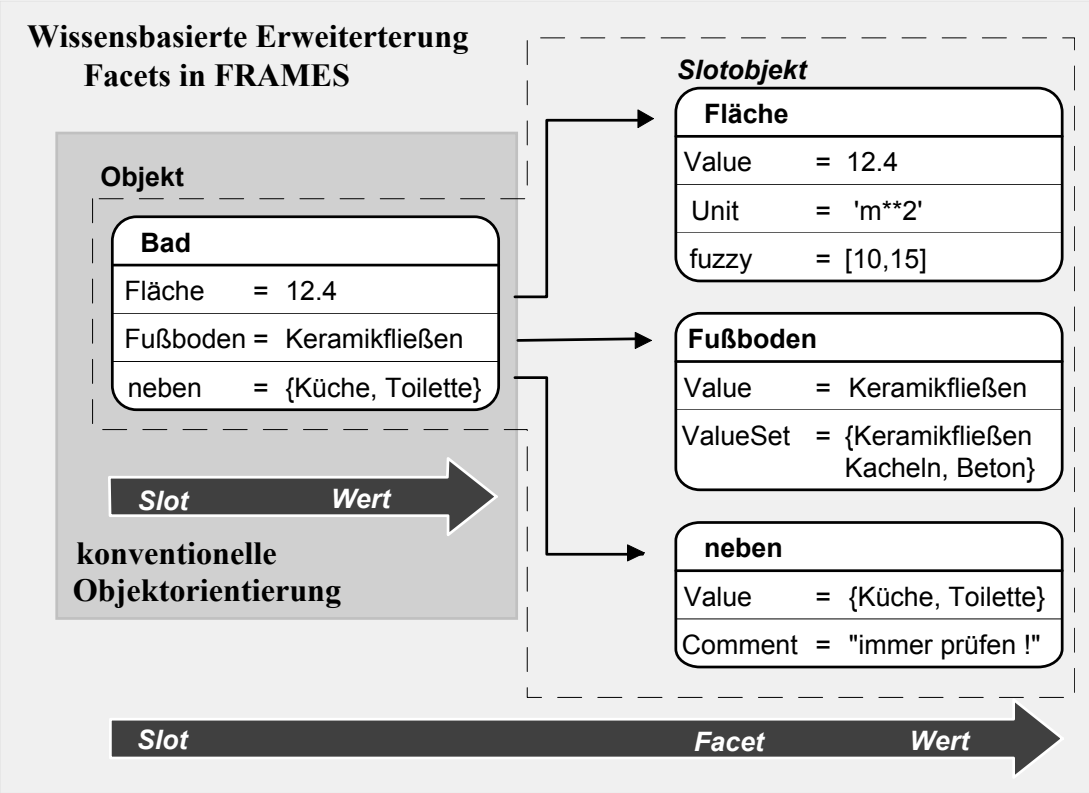


Abb. 6-22 Facetten als Erweiterung des Attribut-/ Relationenkonzeptes

Grundsätzlich ließe sich die rekursive Untersetzung von Informationen beliebig fortsetzen. Dies rührt letztlich aus der Erkenntnis von GÖDEL, daß formale Axiomensysteme unvollständig sind. Die Terminale jeglicher Beschreibung beruhen auf Definitionen, die im Axiomensystem nicht erklärt sind. Fügt man sie hinzu, handelt es sich nicht mehr um Terminale. Dies gilt wiederum rekursiv für die Terminale der Erklärung selbst.
Das Facettenkonzept bildet sowohl als Basis einer Kommunikation zwischen Agenten im Planungsprozeß als auch für die Entwicklung einer technischen Unterstützung in der beschriebenen Form eine gute Basis. Sie entspricht der in praxi beobachtbaren Strukturierungstiefe von Informationen.

6.3 Dokumente – Informale Erweiterung von Modellen

Attribute, Relationen und Methoden sind formalisierte Beschreibungen von Objekten. Sie lassen sich auf genau die deskriptiven Elementtypen oder deren Ableitungen zurückführen, die in einem System jeweils aktuell bekannt sind. Im Rahmen der a priori Formalisierung erfolgt die Fixierung der Syntax und implizit auch der Semantik solcher Attribute und Relationen. Eine derartige Fixierung widerspiegelt die intendierte Modellinterpretation einer Gruppe von Entwicklern und friert dieses zeitlich ein. Für dynamische, nutzeradaptierbare Bauwerks- und Domänenmodelle gelingt diese Strukturfixierung, d.h. Formalisierung je nach angestrebten Konsensgrad, nur partiell.

Die verschiedenen Grade der Formalisierung deskriptiver Modellelemente sind:

(**A**) *Festlegung der Semantik und der Syntax gemäß des Modellierungsparadigmas*
Dies gilt für Merkmale mit einem hohen Konsensgrad innerhalb einer Domäne. Unschärfe ist insbesondere bei numerischen Daten möglich. Solche Merkmale sind Grundlage für Algorithmierbarkeit.

(**B**) *Festlegung der Syntax gemäß des Modellierungsparadigmas*
Dies gilt für Merkmale mittleren oder geringen Konsensgrades und einfacher Struktur. Sie können zur Laufzeit definiert und verwendet werden und müssen durch Nutzer beherrschbar sein. Das Informationsretrieval kann mit standardisierten Such- und Abfragemethoden erfolgen (SQL, OMQL [RAN92]). Abfrageergebnisse lassen sich jedoch auch in Formelauswertern verwenden (Taschenrechnerfunktion, Tabellenkalkulation).

(**C**) *Syntax nicht gemäß des Modellierungsparadigmas – Festlegung eines Medientyps, bzw. einer Dokumentenart*
Dies gilt für Merkmale von (zu) hoher Komplexität, großer Unschärfe und geringen bis mittleren Konsensgrad. Es lassen sich Zusammenhänge beschreiben, die nicht Teil des Domänenmodells sind. Das bedeutet einen völligen Verzicht auf 'berechnende' Maschinenauswertbarkeit. Die Beschreibung erfolgt unter Verwendung von Kommunikationsmitteln, wie natürliche Sprache in akustischer oder textueller Form, Zeichnungen oder Bildern. Dies ist mittels standardisierter Dokumentenformate realisierbar.

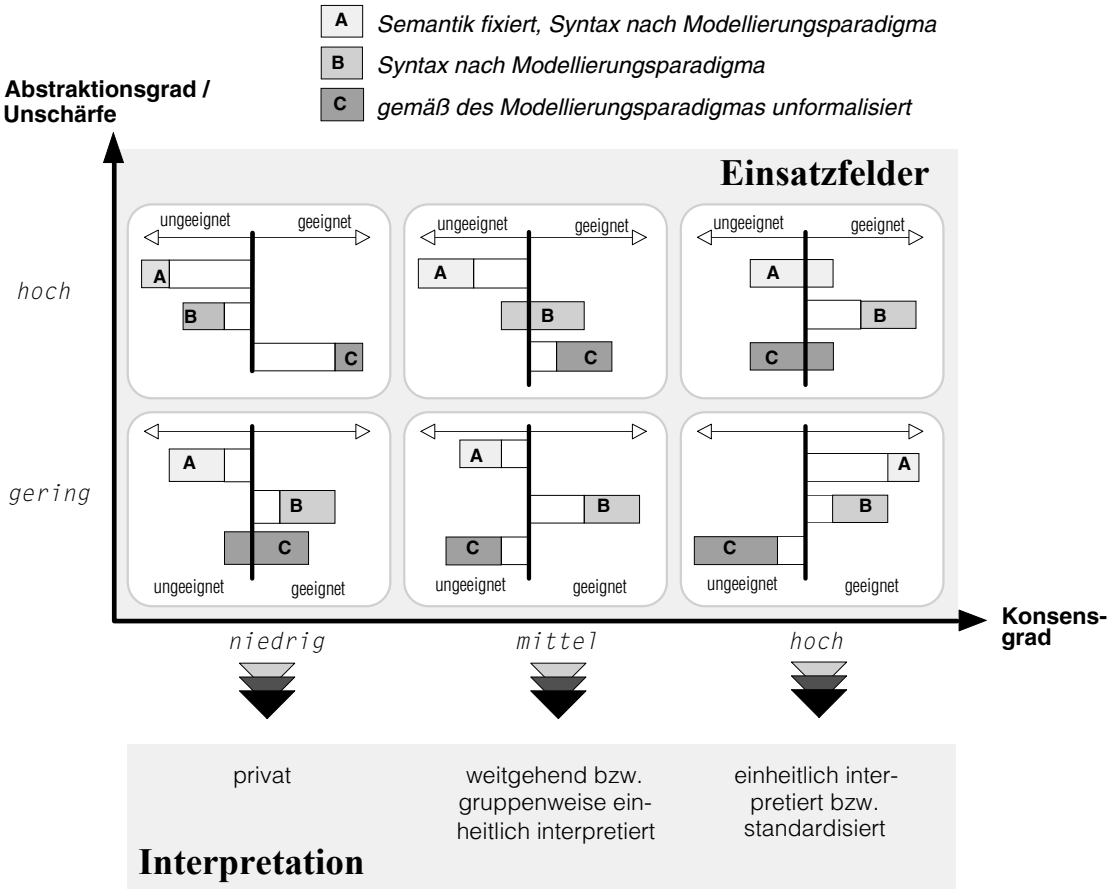


Abb. 6-23 Anwendungsgebiet der Formalisierungstypen deskriptiver Elemente

In einer Büroumgebung umfaßt ein Projekt alle mit ihm assoziierten Vorgänge und Dokumente als Repräsentaten dieser Vorgänge. Gerade in frühen, konzipierenden Entwurfsphasen handelt es sich häufig um schwach strukturierte, unscharfe Daten. Es wird nicht gelingen, alle zur Projektbeschreibung benötigten Daten oder alle mit einer Domäne assoziierten Informationen in einem geschlossenen formalen Modellen vorauszudenken. Die Speicherung aller nicht formalisierbaren Informationen sollte sich daher natürlicher Kommunikationsmedien bedienen.
Neben den menschgemäßen Formen wie, Bild, Ton, Text, können auch speziell technische Repräsentationsmedien Verwendung finden, die keine spezifische Anwendungssemantik tragen und als solche generisch sind. Zu nennen wären hier beispielsweise Zeichnungen (etwa im DXF-Format), Tabellen (in Tabellenkalkulations- oder Standarddatenbanksystemen), anwendungsneutrale Geometriemodelle.
Sind in einer Entwurfsumgebung eine Vielzahl solcher Dokumentenarten unter Einschluß natürlicher Repräsentationsformen verfügbar, spricht man von einem *Multimedialen System*. Technisch realisiert wird dies (z.B. im WorldWideWeb) durch Dokumente, die einem bestimmten Medium zugeordnet sind, in dem sie einer genormten (Datei-) Spezifikation genügen. Eine geeignete Serverapplikation liefert dann die Fähigkeit zur Betrachtung oder Bearbeitung von Dokumenten.
Da Dokumente im allgemeinen keine Entsprechungen für Objekte, sondern nur mit ihnen assoziierte beschreibende Elemente sind, kann man sie in das System der strukturell-syntaktischen Klassifikation von Attributobjekten einordnen. Eine semantische, möglicherweise nutzerspezifische Klassifikation kann diese Basisklassifikation erweitern. (siehe Abb. 6-24)

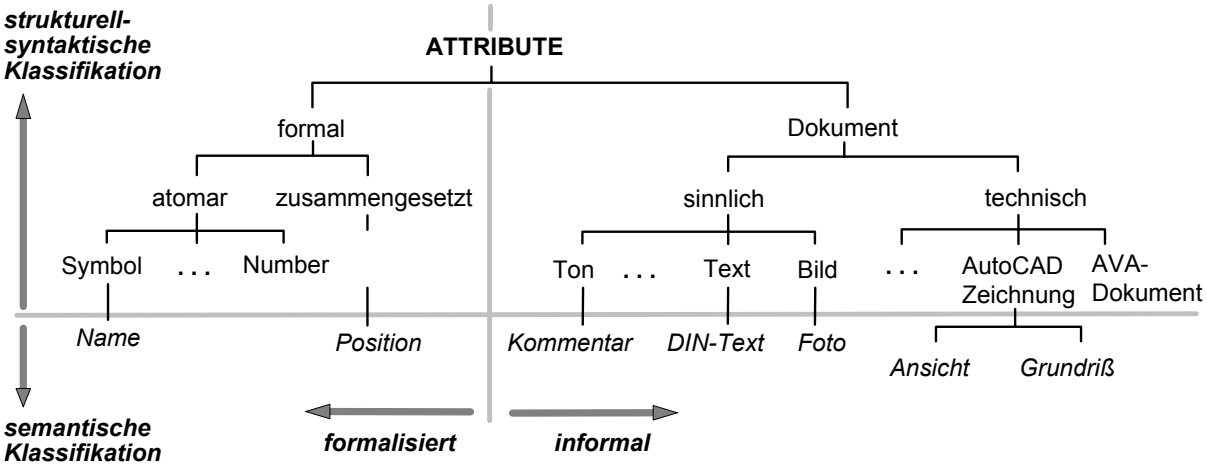


Abb. 6-24 Dokumente als Erweiterung einer Taxonomie von Attributobjekten

Wenn zwischen Dokumenten ein komplexes Beziehungsnetz besteht, werden solche Dokumentensammlung als '*hyper*-medial' bezeichnet**,** wobei Dokumente die Knoten und Verweise (*Links)* die Kanten des Netzgraph darstellen. Links bilden ein den Relationen analoges Beschreibungsmittel. Mit ihnen wird ein navigierender Retrievalprozeß ermöglicht.
Domänen- oder nutzerspezifische Links lassen sich wiederum als Erweiterungen der formalstrukturellen Relationenhierarchie auffassen. Vorteil dieser Betrachtungsweise ist, daß nichtformalisierte Informationen (Dokumente) mittels derselben Strukturierung in die formalen CAD-Modelle eingebunden werden können. Es gibt keinen paradigmatischen Bruch zwischen formalen und informalen Modellinformationen.

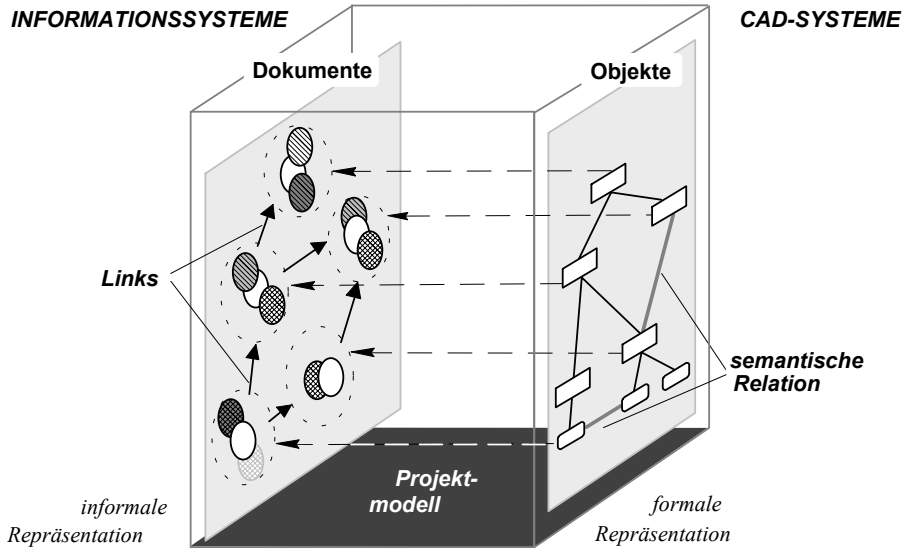


Abb. 6-25 Dokumente als informale Erweiterung von Bauwerks- bzw. Domänenmodellen

In Dokumentensammlungen und in der formalen Modellbeschreibung läßt sich dann mit denselben Mitteln navigieren und zugreifen.
Erzeugt der Architekt Modellelemente eines abstrakten Typs, werden mental alle Informationen, die mit diesem Typus assoziiert sind, aktiviert, wie z.B. DIN-Texte, akustische Memos, technische Dokumente und insbesondere auch Fälle. Durch das holistische Aktivieren eines ganzen assoziativen Umfeldes sind für ihn alle diese Informationen memorierend schnell zugreifbar, wie psychologische Experimente zeigen. Die Verfahrensweise alle, d.h. auch außerhalb des Modellierungsparadigmas liegenden Informationen über Relationen und gleichwertige Links verfügbar zu machen, bildet die natürliche, assoziative Arbeitsweise gut nach.
Sie stützt sich auf das erkenntnispsychologische Modell der 'knowledge chunks' und die auf diesem Modell beruhenden semantischen Netze als Speicherform sowie auf neurophysiologische Modelle der Informationsverarbeitung (assoziatives Gedächtnis). Projekte wie 'CYC'[22] [SCH94] versuchen, große Mengen von Alltagswissen in derartigen schwach strukturierten Modellen zu vergegenständlichen. Die Darstellung eines Begriffs ist dort vergleichsweise simpel. Das meiste Wissen wird in Relationen zwischen den Begriffen repräsentiert.

Mit der beschriebenen Art der Informationsintegration läßt sich ganz wesentlich zur Büroautomation beitragen, da die Möglichkeit besteht, alle elektronisch erfaßbaren Informationen in ein gemeinsames Modell einzubeziehen. Es sei aber ausdrücklich darauf hingewiesen, daß damit die Subjektivität und Kontextabhängigkeit der Informationen sowie ihrer Strukturierung nicht aufgehoben wird. Welche Disziplin und welchen Aufwand der Aufbau und die Pflege solcher komplexer Informationsverwaltungssysteme erfordert, zeigt letztlich der Wildwuchs in Systemen wie des WorldWideWeb's. Der Begriff 'Lost in Hyperspace' [NIE92] beschreibt die Schwierigkeit, dort ein zielgerichtetes Informationsretrieval zu betreiben.
In einer kommerziellen Umgebung steht der Begriff des Dokuments synonym für vertragliche Unterlagen. Das Beglaubigen und Autorisieren von elektronischen Dokumenten stellt jedoch

[22] 'CYC'-Projekt dient der Abbildung von Alltagswissen in einer 'chunk'-'link'-Struktur, um es einer breiten Zahl von Anwendungen bis hin zu Betriebssystemen verfügbar zu machen. Es wird von DEC, MicroSoft, Apple, Boing, AMD u.a. unterstützt. Das Projekt läuft seit 1984 und erfaßte 1989 1Mio. Aussagen über 30000 Objekte.

nach wie vor ein technisches Problem dar. Dies betrifft auch das kooperative Arbeiten in komplexen Informationsbeständen.

6.4 Unschärfe – Qualität von Entwurfsobjekten

Eines der Probleme bei der Nutzung bestehender CAD-Systeme in frühen Entwurfsphasen ist der Zwang zur Exaktheit bei Eingaben. Dies wird zwar durch die Möglichkeiten des grafischen Interagierens für die Eingabe geometrischer Attribute gemildert, deren Darstellung in der Genauigkeit der internen Gleitkommadarstellung entspricht in keiner Weise dem Charakter von Entwurfsobjekten, wie sie in frühen Phasen bearbeitet werden. Unsicherheit, Unschärfe, Vagheit und Unvollständigkeit tritt in vielfältiger Weise im Entwurfsprozeß auf und kann hier nicht erschöpfend behandelt werden[23] . Für den Architekten bringen insbesondere seine gewohnten grafischen, skizzenhaften Ausdrucksmittel Unschärfe zum Ausdruck. Unter Unschärfe versteht er häufig 'vage' beschriebene geometrische Eigenschaften von Modellobjekten. Wie die Skizzentechniken sind auch verschiedene Formen von Unschärfe an bestimmte Entwurfszustände bzw. Entwurfsphasen gebunden.
Der Begriff der Unschärfe (engl. fuzziness) ist seit einer Veröffentlichung von ZADEH von 1965 ([ZIM92]) in der Diskussion und bezog sich dort auf die vage, nicht sicher angebbare Zugehörigkeit eines Objektes zu einer Menge. Heute wird dieser Begriff jedoch als Synonym für ein Spektrum von Formen der Vagheit verwendet. Mehrere Theorien befassen sich mit Vagheit und ihrer Abbildung (fuzzy sets, fuzzy logic, modale Logiken, Shafer-Demster Theorie, qualitatives Schließen, probabilistisches Schließen u.a.m., siehe etwa [ZIM93]). Diese Theorien sind oft umstritten und unvollständig, in ihrem Begriffsapparat schwierig und der abzubildenden Realität oft nicht angemessen. Außerdem spiegeln sie jeweils nur einen Aspekt der Vagheit wieder.

6.4.1 Klassifikation von Unschärfe

Analog der Methodik zur Untersuchung des architektonischen Entwurfsprozesses läßt sich Vagheit einerseits dem Modell des Entwurfsgegenstandes zuordnen, andererseits dem Prozeß der Erstellung dieses Modells. Zur Untersuchung *handlungsbezogener Unschärfe* liegen kaum Erkenntnisse vor, sie werden widergespiegelt durch Entwurfsaktionen des Typs 'Steuerung'.
Bei der Bewertung von Entwürfen bzw. deren Varianten in einem Evaluierungsprozeß ist mit der *zielbezogenen Unschärfe* eine weitere Form von Vagheit im Planungsprozeß beobachtbar. Eine eindeutige Bewertung und Selektion einer Entwurfsvariante ist auf Grund der i. allg. mehrkriteriellen Ziele nicht möglich, zumal die einzelnen Zielkriterien durch Verrechnung unscharfer Modellparameter erstellt werden, bzw. die Berechnungsverfahren selbst approximativ sind. Die Strategie einer CAD-Stützung kann hier nur eine umfassende Informationsaufbereitung und Verdichtung sein, wie sie beispielsweise Erfolgsspinnen für numerische Zielkriterien darstellen ([STE90], [GYA94]) oder Visualisierungen der Gestalt zur ästhetischen Bewertung.

Für die Modellobjekte selbst läßt sich Unschärfe untergliedern in

- *Syntaktische Unschärfe*, d.h. Modellinformationen, die im gewählten Metamodell nicht zu formalisieren sind oder deren formalisierte Abbildung zu aufwendig erscheint. Eine

[23] siehe etwa [RED95b], einen Überblick im Bezug auf CAD-Modelle gibt HUPFER [HUP95]).

Beschreibung erfolgt in der aus Modellsicht unstrukturierten Form der natürlichen Kommunikationsmedien oder aber generischer technischer Medien.

- *Semantische Unschärfe*, d.h. Formen von Unschärfe, die nicht durch den Charakter des aktuellen Modells selbst begründet sind und die in ihm dargestellt werden können und sollen.

Modellimmanente Unschärfe entsteht durch die Verkürzungseigenschaft der Modellbildung selbst. Da im Planungsprozeß vollständige Planungsunterlagen geschaffen werden sollen, sind folglich Objekte früher Phasen in diesem Sinne immer unscharf. In den Modellwelten der einzelnen Entwurfsphasen oder Ingenieurgewerke können diese Modelle jedoch jeweils scharf, d.h. vollständig und exakt beschrieben sein. In diesem Sinne kann Unschärfe oder Vagheit wie folgt definiert werden: *Unschärfe ist der informationelle Abstand zum Grad der vollständigen, exakten Beschreibung, der in einer bestimmten Modellwelt erzielt werden kann.*

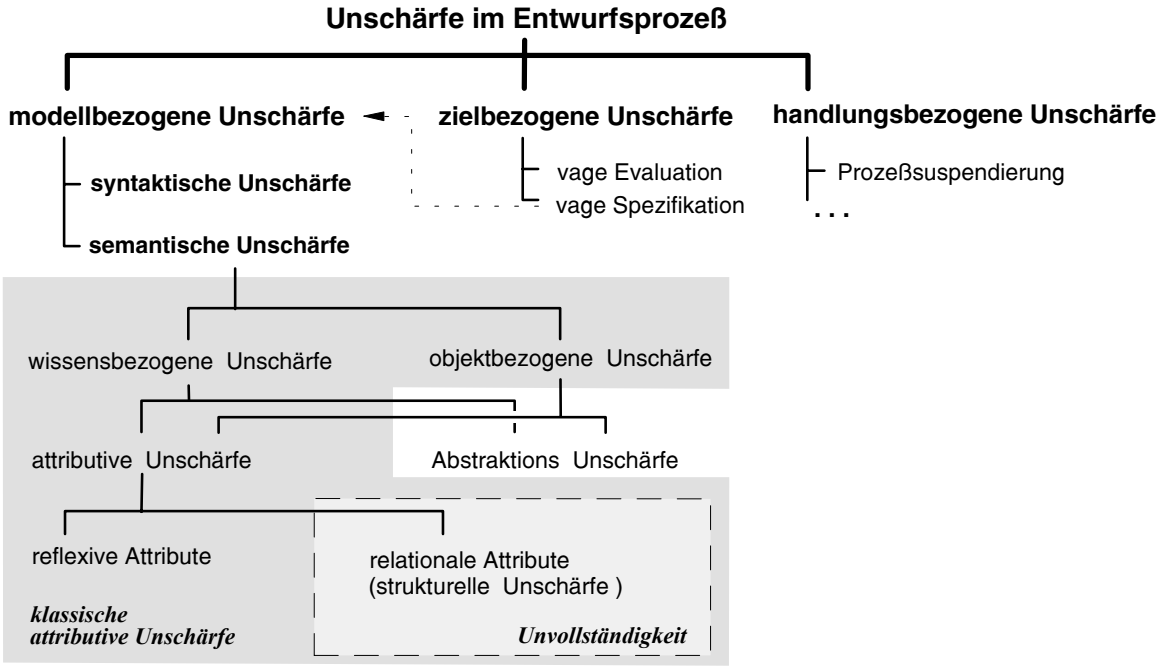


Abb. 6-26 Formen von Unschärfe im Entwurf

6.4.2 Attributive Unschärfe

Die klassische Form der *objektbezogenen Unschärfe* bezieht sich auf die unscharfe Ausprägung von (numerischen) Attributen. Ausgehend von der Erkenntnis, daß bei der Beschreibung der realen Welt häufig der Fall auftritt, daß Erscheinungen nicht eindeutig einer Klasse zuzuordnen sind, ersetzte ZADEH diese binäre Zugehörigkeit durch einen 'Grad' der Zugehörigkeit in Form einer Zugehörigkeitsfunktion $p\,(O, M)$ eines Objektes o_i zur Menge M von

$$\textit{nicht zugehörig} \;=\; 0 \le p(o_i, M) \le 1 \;=\; \textit{vollständig zugehörig}$$

und baute auf dieser Funktion eine neue Logik, die *'fuzzy logic'* auf. Entsprechend beschriebene Mengen sind '*fuzzy sets*'.

Im Kontext von CAD-Modellen ist z.B. ein exakt beschriebener Punkt $P_{NW} = (x, y)$ einer Menge M_{RO} von Punkten 'rechts-oben' zugehörig. Die Menge ist unscharf durch eine Zuge-

hörigkeitsfunktion $p((x,y),M_{RO})$ gegeben (Abb. 6-27 a). Im Entwurf tritt hingegen oft der analoge Fall auf, daß man Angaben zu Attributen von Objekten nur unscharf oder qualitativ machen kann oder will. So kann für die Gebäudeecke P_{NW} die Lage 'rechts oben' spezifiziert sein. (Abb. 6-27 b)

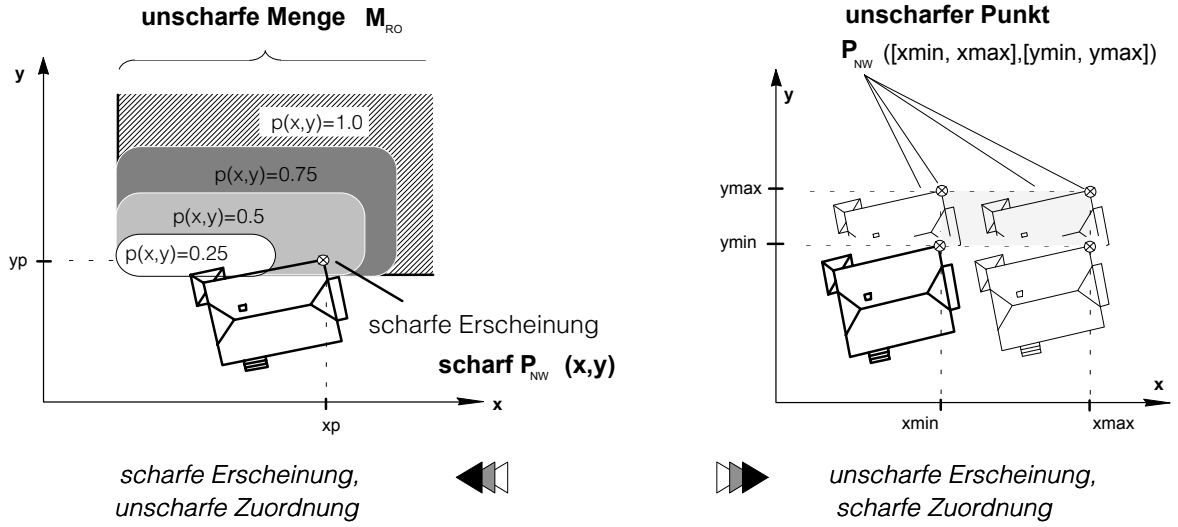


Abb. 6-27 a, b Unscharfe Positionierung eines Gebäudegrundrisses

Obwohl diese unscharfen Werte im Zuge des Entwurfsprozesses durch die zur Bauwerkserstellung notwendigen exakten Werte substituiert werden, gewinnen sie in späteren Entwurfsphasen wieder Bedeutung. Soll beispielsweise später ein Fahrweg über das Grundstück führen, so kann die Frage: „Welche Bauwerke sind nahe des Weges" auftreten. Bei der alleinigen Verwendung 'exakter' Modelle könnten Bruchteile eines Millimeters darüber entscheiden, daß diese Frage für 'Maiers_Haus' verneint würde. Dieses Verhalten entspricht in keiner Weise der eines menschlichen Planers.
Die Art und Semantik der für CAD-Modellierungsprozesse angewandten Formen von Vagheit differieren beträchtlich. Sie hängen wesentlich vom Attributtyp der exakten Beschreibung ab. Eine ungenaue Werteintragung bei reflexiven Attributen entspricht am ehesten der konventionellen Vorstellung von Ungenauigkeit oder Toleranz. Für ein symbolisches Attribut kann man z.B. ausdrücken, daß bei

Bad die Wandfarbe nicht genau grün ist, sondern eine aus {gelb, grün, blau}.
⇧ ⇧ ⇧ ⇧
Objekt *Attribut* *scharfer Werteintrag* *unscharfer Werteintrag*

Der Wert von Attributen kann aber auch linguistischer Natur sein (z.B. 'kalt', 'fest'). Nicht zu verwechseln ist diese Unschärfe mit dem symbolischen Wert von Attributen (gut, schön, süßlich etc.) für die sich prinzipiell keine (sinnvolle) numerische Interpretation finden läßt. Dies betrifft vor allem Sinneswahrnehmungen und Empfindungen.

Für viele Auswertungen über derartige Attribute werden exakte Werte benötigt, die entweder explizit verfügbar sind oder durch eine geeignete Defuzzifizierung erstellt werden. Unschärfe kann auch als Wahrscheinlichkeiten für mögliche Wertebelegungen interpretiert werden. Eine Variante zur fünfstufigen Abbildung von Unschärfe zeigt Abb. 6-28 a und b.

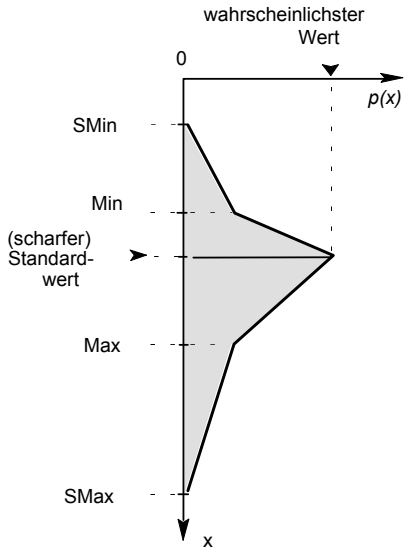


Abb. 6-28 a Verteilung vager Belegungen numerischer Attribute

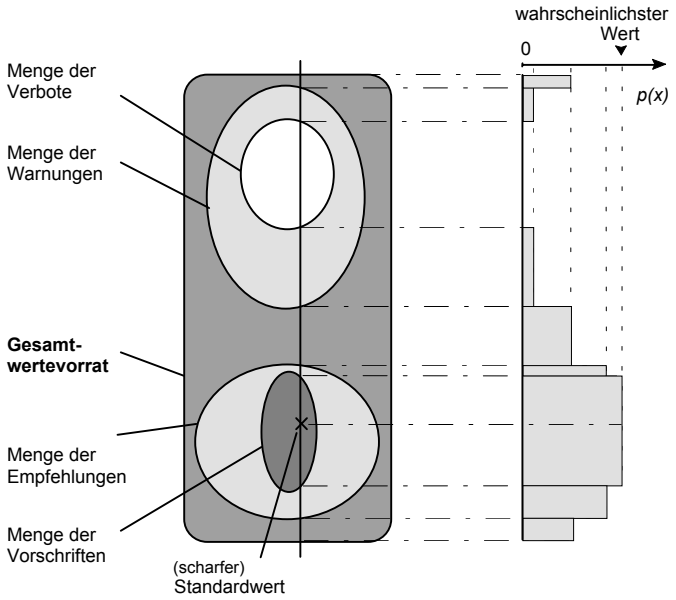


Abb. 6-28 b Venn-Diagramm der vagen Belegungen symbolischer Attribute

Ungewißheiten auf attributiver Ebene betonen dagegen einen anderen Aspekt von Vagheit. Gemeint ist hier die (Un-) Zuverlässigkeit von an sich exakten Wertangaben. Klassisches Beispiel ist die Festigkeitsangabe bei Werkstoffen (z.B. 5% Quantil bei Beton, d.h. mit 95%iger Sicherheit ist die Druckfestigkeit von Beton größer gleich dem angegebenen Wert). Der Wert der Ungewißheit kann selbst wieder unscharf werden, die Rekursion ließe sich beliebig fortsetzen. Ein Beispiel: Der Raum ist sicherlich (linguistischer Gewißheitswert) sehr hell (linguistischer Attributwert).

Unvollständigkeit als stärkste Form von Vagheit tritt in besonderem Maße in frühen Entwurfsphasen auf. Für Attribute ist hierunter sowohl das Fehlen des Wertes eines Attributes zu verstehen, als auch das völlige Fehlen des Attributes selbst.

6.4.3 Strukturelle Unschärfe

Strukturelle Aspekte des Bauwerksmodells werden durch Relationen zwischen Objekten abgebildet. Strukturelle Unschärfe von Modellen muß also auf der Ebene von Relationen beschrieben sein. So kann z.B. in einer 1:m Relation die Menge der referenzierten Objekte als Auswahlmenge verstanden werden, d.h. eine Relation existiert nicht zu allen Objekten der Auswahlmenge (scharfe Beschreibung), sondern nur zu einer Teilmenge. Eine weitere Möglichkeit ist die Verwendung abstrakter Objekte, d.h. von Klassenbezeichnern in der Auswahlmenge. In diesem Fall wird davon ausgegangen, daß die Relation zu einem Objekt oder einer Menge von Objekten der angegebenen Klasse existieren soll. Auch auf der Ebene struktureller Unschärfe läßt sich ein fünfstufiges Konzept wie bei reflexiven Attributen umsetzen, es wäre analog zur mengenbasierten Darstellung bei symbolischen Attributen zu gestalten.

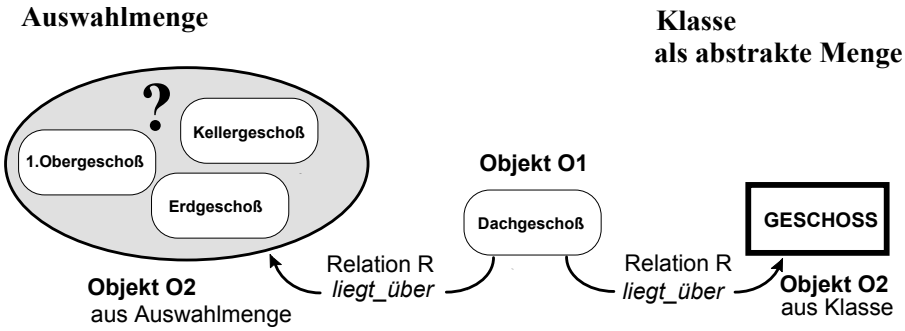


Abb. 6-29 Strukturelle Unschärfe bei Relationen

Auch auf strukturellem Niveau läßt sich Ungewißheit beobachten, wie z.B. die Aussage: „Das Bad liegt wahrscheinlich neben der Küche“ verdeutlicht. Ebenso fallen mit Ungewißheit behaftete Strukturentscheidungen in diese Kategorie, wie etwa: „sicher 5 Geschosse, vielleicht 6 Geschosse“. Am ehesten läßt sie sich als Variantenbildung interpretieren, wie etwa als (wahrscheinlichkeitsbehaftete) Alternativen bei der Untersetzung komplexer Objekte durch Teile.
Unvollständigkeit ist auf struktureller Ebene als das völlige Fehlen referenzierter Objekte zu verstehen, d.h. diese sind als eigenständige, manipulierbare Objekte noch nicht erzeugt worden. So kann man beispielsweise bei einem Objekt ’Dachgeschoß’ angeben, daß es die Relation ’liegt_über’ zu einem anderen Objekt besitzt, welches aber als Referenzobjekt noch nicht im Modell verfügbar ist.

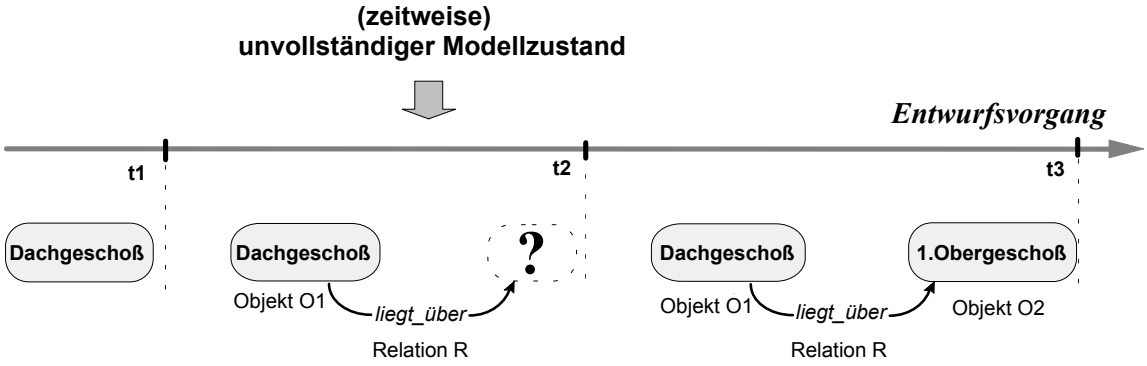


Abb. 6-30 Strukturelle Unvollständigkeit

Alle bisher beschriebenen Formen der Vagheit beziehen sich auf Entwurfsobjekte (Instanzen), mit denen der Architekt seine Entwurfsintensionen modellhaft beschreibt. Sie lassen sich jedoch analog auf das Wissen über Objekte bzw. Mengen von Objekten (Klassen) übertragen. Klassen sind als Generalisierung bzw. Abstraktionen von konkreten Erscheinungen a priori mit Vagheit behaftet. Wenn heute von einer unscharfen Wissensbasis die Rede ist, so bedarf das einer weiteren Untersetzung. In der Praxis sind das i. allg. Systeme, die mit attributiver Unschärfe oder Ungewißheit umgehen können. Falls solche Systeme mehrere Arten von Vagheit verkraften sollen, erfordert dies zusätzliches Metawissen zur Abarbeitungssteuerung, zur Fuzzifizierung/Defuzzifizierung, zur Konfliktlösung sowie eine ausgefeilte Symbolverarbeitung. Die oft benutzte Bezeichnung ‘unvollständiges Wissen’ ist stark mißverständlich, da Wissen, das fehlt, natürlich nicht zur Problemlösung herangezogen werden kann. Gemeint sind hier fehlende oder unvollständige Daten.

6.4.4 Abstraktion als Form der Unschärfe

Die vorn beschriebene Ordnung von Klassen (-wissen) in Form von Taxonomien eignet sich in besonderem Maße zur Abbildung von *Abstraktionsunschärfe*. Wie schon dargelegt wurde, erfolgt Entwerfen mental grundsätzlich top-down, d.h. vom Ganzen zu seinen Teilen, vom Abstrakten zum Konkreten. In frühen Phasen werden Objekte von stärker abstrahierenden Klassen abgeleitet. Sie sind auf Grund des geringeren Wissens, das abstraktere Klassen tragen, unschärfer und sie fixieren weniger Eigenschaften in weniger genauer Weise als solche, die man in späteren Phasen erzeugt. So genügt es möglicherweise anfangs 'Maiers_Haus' als Ausprägung der Klasse 'Wohnbauten' zu erzeugen, da die Merkmale dieser Klassen, wie z.B. Baukosten, Grundfläche, Bauherr, Wohnfläche etc., für eine vorläufige Beschreibung genügen. Mehr Aussagen sind für 'Maiers_Haus' noch nicht verfügbar oder noch nicht erforderlich.

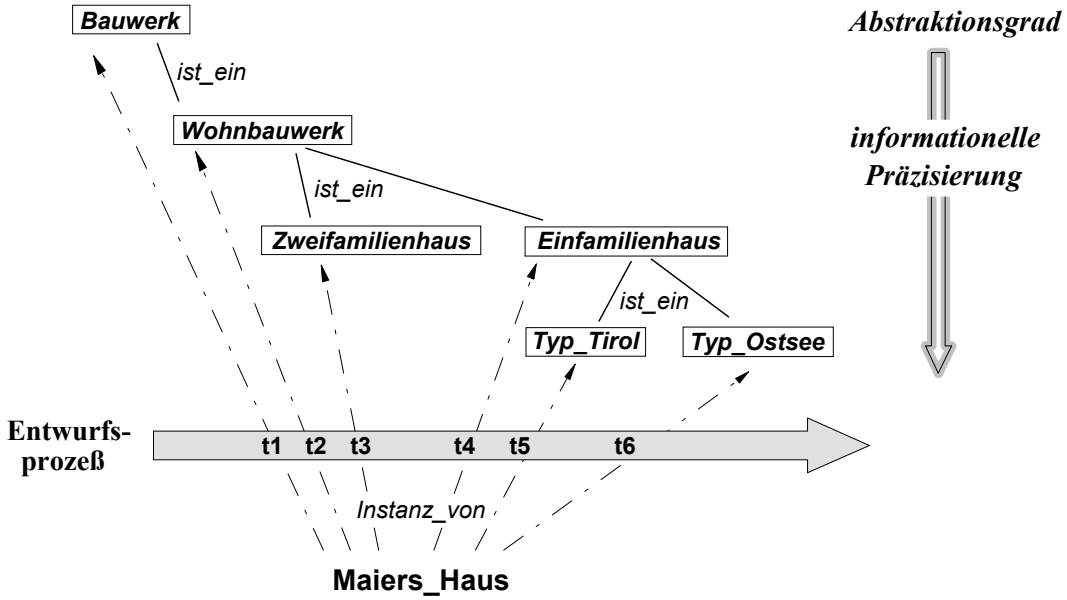


Abb. 6-31 Abstraktionsunschärfe durch Klassifikation

Die pure Existenz der noch abstrakten Modellobjekte gestattet jedoch, weitere Entwurfshandlungen an oder mit ihm zu vollziehen. Im weiteren Verlauf der Entwurfstätigkeit können diese existenten Objekte präzisiert werden, indem sie

- durch genauere Klassifikation genauer beschrieben werden
- Standardannahmen zugeordnet bzw. überschreiben werden, deren Unschärfegrad hängt vom Abstraktionsgrad der Klasse ab (attributive Unschärfe)
- durch Einfügen von Bauteilen verfeinert werden (strukturelle bzw. kompositionelle Unschärfe).

Jede Entwurfsphase verfügt über ein Domänenmodell, dessen Abstraktionsgrad für diese Phase spezifisch ist. Dies spiegelt sich in der Strukturtiefe der hierarchischen Aggregationsrelation wider, die das entsprechende Domänenmodell zuläßt. Ein Modell ist scharf beschrieben, wenn alle 'Blatt-Objekte' der Aufbaustruktur sich in der Domäne nicht mehr durch Teile untersetzen lassen. Gerade beim top-down Entwurf beginnt der Entwurf mit dem Wurzelobjekt der Aggregation, d.h. das Objektmodell ist zu diesem Zeitpunkt sehr unscharf. Gleiches gilt auch beim Übergang zwischen Entwurfsphasen. So terminiert der Entwurfsprozeß in der Vorplanung von Wohngebäuden auf Raumniveau, während für die

Genehmigungsplanung Kenntnisse über konstruktive Objekte fixiert sein müssen, die die Räume bilden. Beim jedem Phasenübergang ist das Entwurfsmodell in der jeweils neuen Modellwelt initial unscharf.

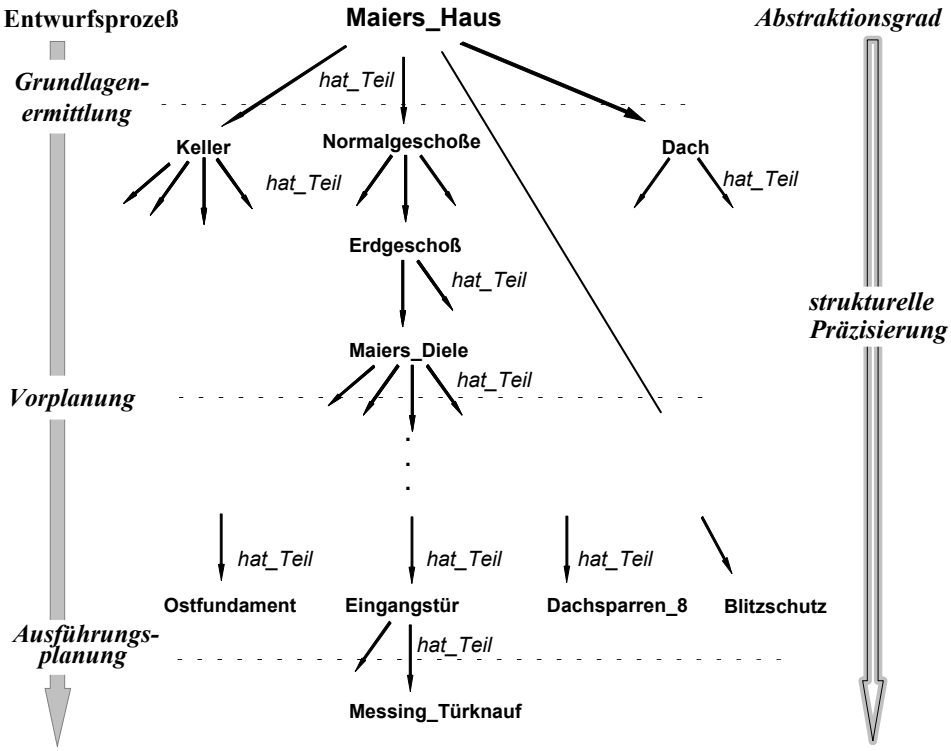


Abb. 6-32 Beispiel einer Kompositionhierarchie

Unserer Umwelt ist voll von künstlichen Objekten, die vorgedacht wurden in einem solchen mit Unschärfen überfrachteten Prozeß. Dasselbe gilt für die Theoriebildung über Unschärfe. KRUSE et.al. [KRU93] führen aus: „Innerhalb der Mathematik wurden fuzzy-Mengen kontrovers diskutiert... (nur) einige wenige seriöse Veröffentlichungen erschienen... Dagegen stehen jedoch viele erfolgreiche Anwendungen".

Wenn man all diese Formen der Vagheit anerkennt und um die Schwierigkeiten weiß, die es macht, solche Art Wissen zu akquirieren, besteht die Gefahr in Skeptizismus oder gar Nihilismus zu verfallen. Hierauf erwidert KURTZ[24] „... wir müssen einige Überzeugungen entwickeln, nach denen wir leben und handeln können. Vielleicht gründen sich unsere Überzeugungen in letzter Konsequenz nur auf Wahrscheinlichkeiten – wir jedenfalls brauchen Erkenntnisse und Wissen als pragmatische Voraussetzung für unser Leben und Handeln in der Welt." (aus [KUR96])]

[24] P.KURTZ, Vertreter des „gemäßigten Skeptizismus", Philosoph an der State Uni. of New York

6.5 Objektorientierung – Modellierungs- und Modellbildungs- paradigma im computergestützten Entwurf

Versteht man Entwerfen als Modellbildungsprozeß, der in der Vorwegnahme imaginierter Artefakte in Modellen besteht, läßt sich der Entwurfsgegenstand sehr gut über das Konzept des '*Objekts*' beschreiben. Reale oder nur in der Vorstellung des Architekten befindliche Artfakte sind die Urbilder, die durch die Modellobjekte widergespiegelt werden. Die Objektorientierung bietet eine stringente Modellierungsmethodik mittels der alle inhaltlich zusammengehörenden Informationen kompakt abgelegt werden können. Durch die Einordnung der Objekte in ein begriffliches System (Taxonomie) wird der Zugriff auf formalisiertes Wissen zur Objektdeskription (Merkmale und deren Werte), zum Objektverhalten (Methoden, Regeln, ...) aber auch auf informales Wissen (Dokumente, alte Projekte ...) möglich. Das Konzept ermöglicht die reflexive Beschreibung von Objekten (Attribute) sowie der Struktur komplexer, rekursiv aus Objekten aufgebauter Modelle (Relationen, insbesondere die Aggregation). Formen der Unschärfe, soweit sie Objekten zugeordnet werden können, lassen sich in dem Konzept homogen erklären. Dies betrifft neben der 'konventionellen' Unschärfe von Attributen auch die strukturelle Unschärfe.
Ein wechselnder Abstraktionsgrad von Objekten, wie er im Entwurfsprozeß auftritt, ist ebenfalls einfach realisierbar. Dies erfolgt durch den Wechsel der Klasse, von der Modellobjekte abgeleitet wurden oder durch Ändern der Betrachtungsebene im Aggregationsgraphen.

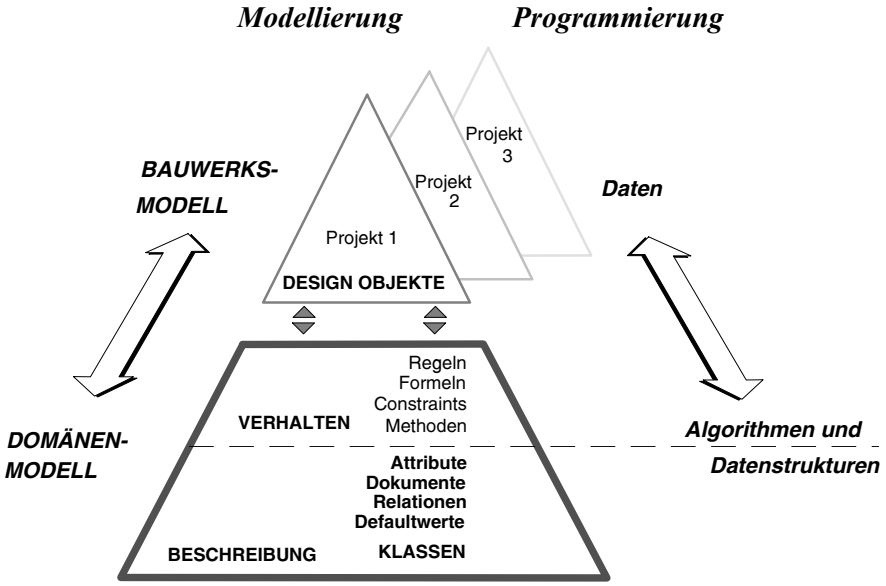


Abb. 6-33 Bauwerksmodelle als Extensionen von Domänenmodellen

Auf der Basis von Objekten läßt sich der Entwurfsprozeß durchgängig unterstützen. So lange das Urbild existiert, sollte auch sein Modellabbild existieren. Das Objekt hat, einmal kreiert, Bestand über die Entwurfs- und Nutzungsphasen. In den semantisch verschiedenen Kontexten der Entwurfsaspekte ändern die Objekte zwar ihre Darstellung und ihren Informationsgehalt - was bleibt, ist das Objekt an sich! Stellt man (soweit als möglich) den Modellobjekten noch das zum Erzeugungszeitpunkt gültige Klassenwissen zur Seite, so konstituiert diese Einheit aus generalisiertem Wissen und konkreter Ausprägung ein Bauwerksmodell, dessen Interpretation über einen langen Zeitraum und unter vielfäligen Gesichtspunkten möglich sein sollte.

Der besonderer Vorteil dieser Technologie ist, daß sie das Bilden von Bauwerksmodellen als Entwurfstätigkeit und das Bilden von Domänenmodellen als Wissenserwerbsprozeß in einem homogenen System erklärt. Bauwerksmodelle sind als solche Extensionen von Domänenmodellen (Abb. 6-33).
Mit dem Prinzip der Objektorientiertheit scheint darüber hinaus die Klammer gefunden zu sein, die die Phasen Softwareentwicklung und Softwarenutzung verbindet. Softwareentwickler und Nutzer können (oder zumindest könnten) für ihre Arbeit in gleicher Weise Begriffe der Domäne auf semantisch hohem Niveau nutzen. Durch die Nutzung des deklarativen Charakters der OOP ist es bei entsprechend vorbereiteten Modellverwaltungssystemen möglich, Teile des Entwicklungs- und Wartungsaufwandes zum Nutzer zu verlagern. Damit ist die Flexibilität gegeben, privates 'know how' in den Systemen zu vergegenständlichen und die Systeme ständig anpassen und weiterentwickeln zu können.

Bedingung der Abbildbarkeit von Domäneninformationen ist wiederum ein Modell, das die vorgestellten Elemente des Domänenwissens zu repräsentieren vermag. In Abhängigkeit der erreichbaren Formalisierung und Standardisierung lassen sich so CAD-Tools implementieren, um

- Entwurfsvorgänge zu automatisieren, wo dies sinnvoll ist (sehr hoher Formalisierungsgrad),
- Entwurfsvorgänge zu assistieren, wo dies möglich ist (hoher Formalisierungsgrad),
- oder als Ergebnis von Entwurfsvorgängen eben nur Bauwerke langlebig, gut interpretierbar zu beschreiben (niedriger Formalisierungsgrad).

Das benötigte Metamodell ist weiterhin eine geeignete Basis, um die vorgeschlagenen generischen, am kognitiven Entwurfsprozeßmodell orientierten Basistätigkeiten zu definieren und sie gemeinsam mit dem Metamodell verfügbar zu machen. Die Einheit von Metamodell zur Beschreibung von Bauwerks- und Domänenmodell sowie einem zugehörigen Prozeßmodell gestattet, daß der Nutzer nicht nur adäquate Modellelemente vorfindet, sondern daß auch der externe Modellaufbau analog zum imaginierten Modell erfolgen kann.

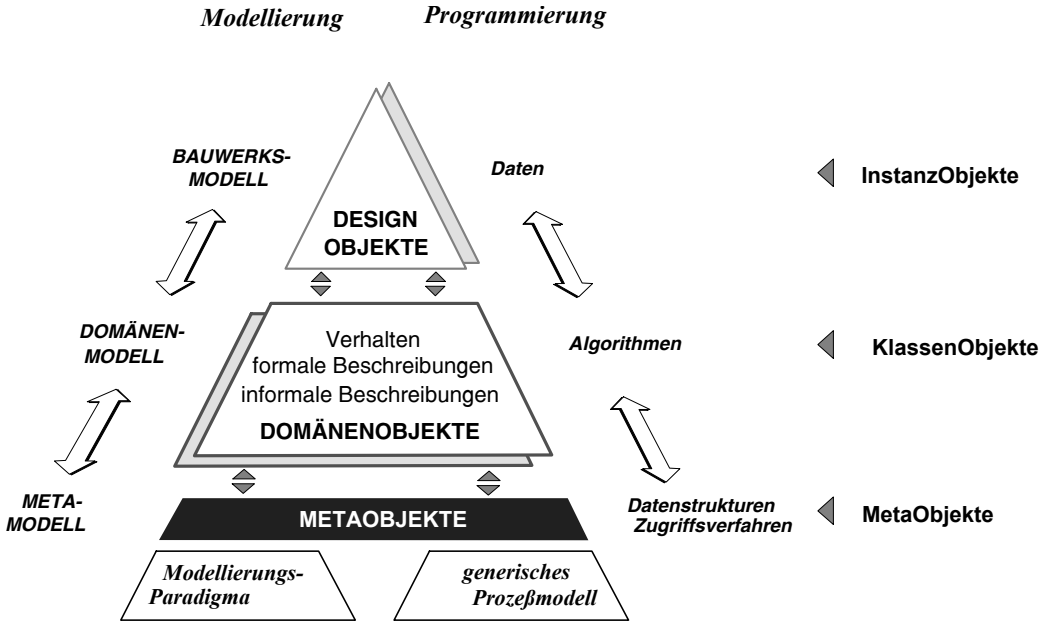


Abb. 6-34 Domänenmodelle als Extensionen eines Metamodells

Die folgenden Stichpunkte formulieren noch einmal zusammenfassend die Anforderungen an CAD-Tools bzw. Systeme sowie für die ihnen zugrundeliegenden Modellverwaltungssysteme (MVS) soweit sie aus den Ausführungen zur Objektorientierung folgen.

(1) Das Erzeugen symbolisch-begrifflicher Repräsentationen in Form von Modellobjekten, die in ihrer Gesamtheit ein Bauwerksmodell konstituieren, ist eine für die Bauwerksplanung geeignete Modellierungstechnologie.

(2) Symbolisch-begriffliche Repräsentationen sind eine geeignete Grundlage zur Entwicklung von CAD-Systemen. Die Objektorientierung vermag so die Phasen Systemnutzung und Systementwicklung zu vereinheitlichen. Während sich OO-Entwicklungsumgebungen weitgehend durchgesetzt haben, sind echt objektorientierte CAD-Systeme jedoch nicht verfügbar.

(3) Die bereits in frühen Entwurfsphasen fixierte Aggregationsstruktur von Bauwerken ist von großer Bedeutung sowohl für die Modellorganisation als auch zur Steuerung des Entwurfsprozesses. Die explizite Unterstützung dieser Relation ist sowohl durch die CAD-Tools als auch durch die MVS selbst zu gewährleisten.

(4) In Abhängigkeit von der Entwurfsphase sind Modellobjekte in verschiedener Weise Träger von Unschärfe. Diese modellbezogene Unschärfe gilt es soweit als möglich zu unterstützen.

(5) Die Klassenhierarchie, aus der konkrete Modellelemente abgeleitet werden, eignet sich zur Abbildung von Wissen. Um dieses Wissen durch den Nutzer zugreifbar, änderbar und erweiterbar zu machen, sind Klassenobjekte erforderlich, die zur Laufzeit verfügbar sind. Basis der Beschreibung von Wissenselementen ist ein Metamodell, das durch das Modellverwaltungssystem realisiert wird.

(6) Insbesondere für den Bauwerksentwurf gelingt es nicht, alle relevanten Daten bzw. alles Wissen zu formalisieren. Alle nichtformalisierten Informationen sollten in Form von, aus Modellsicht unstrukturierten, Dokumenten in das Modell eingebracht werden können. Um diese Informationen kontextsensitiv verfügbar zu machen, dient zu dessen Strukturierung die Taxonomie.

7. SYSTEMKONZEPT PREPLAN

7.1 Systemphilosophie

PREPLAN ist ein Konzept zur Strukturierung eines Sets von Tools zur Unterstützung des architektonischen Entwurfs insbesondere in frühen Phasen. Auf deren besondere Bedeutung im Prozeß des Planen, Herstellen und Bewirtschaften von Gebäuden wurde bereits eingegangen. Die Tools formen ein System von CAAD-Werkzeugen, das als Ganzes den Gedanken des 'design assistant' umsetzt ([AYE91]). Das System will den Nutzer bei der systematischen Erfassung seiner Entwurfsidee in einem Modell unterstützen sowie dieses Modell soweit als möglich plausibel halten.
Die Module des Systems sind zur Unterstützung jeweils nur eines Entwurfsaspekts konzipiert. Die Art und das Vorgehen beim Entwerfen/Modellieren ist abhängig vom gewählten Modellierungsparadigma, welches hier einheitlich die *Objektorientierung* ist. Die Einzeltools basieren auf demselben Vorrat von Handlungstypen, wie sie einerseits aus dem kognitiven Prozeßmodell und andererseits dem Modellierungsparadigma folgen. Funktionalität und Nutzungstechnologie können hierdurch vom betrachteten Aspekt des Entwerfens, der unterstützten Entwurfsphase sowie der gewählten Repräsentation entkoppelt und für alle Tools homogen gestaltet werden.

Das Konzept PREPLAN versucht kognitive und pragmatische Sichten zu vereinen, indem

- *Werkzeuge für frühe Phasen des Entwurfes konzipiert werden*

Späte Phasen des pragmatisch gegliederten Planungsprozesses sind bereits gut unterstützt (oder doch unterstützbar). Die Spezifik von Objektmodellen früher Phasen wie Unschärfe und Abstraktion finden Beachtung.

- *die einzelnen Werkzeuge ein durchgängiges System bilden*

Die pragmatisch gebildete Phase 'früher Entwurf' (LP (1), 2, (3) HOAI) wird durch Werkzeuge so unterstützt, daß alle Aspekte des architektonischen Entwerfens Beachtung finden.

- *das kognitive Prozeßmodell in allen Einzelwerkzeugen realisiert wird*

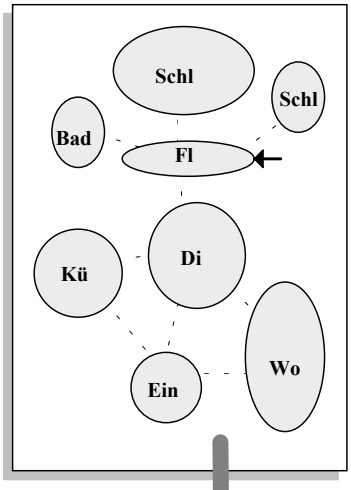
Basis hierzu ist eine Modellstrukturierung (Metamodell), die in allen Aspekten (Funktion, Form, Konstruktion) einheitlich ist. Auf diesem Metamodell wird für jeden Modul der gleiche Satz atomarer Designaktionen definiert, so wie er aus dem kognitiven Prozeßmodell folgt.

- *ein im einzelnen nicht vorhersagbarer Prozeßverlauf akzeptiert wird*

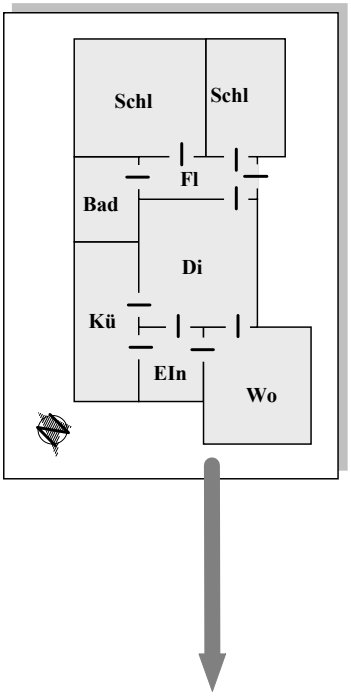
Dies wird durch die nicht zwangsläufig sequentielle Nutzung der Tools erreicht, die die einzelnen Aspekte realisieren. Weiterhin wird eine opportunistische Designstrategie durch die explizite Unterstützung einer Aufbaustruktur möglich, deren Verwaltung wiederum in allen Tools durch atomare Entwurfsaktionen vereinheitlicht wird.

Wesentliches Anliegen der Tools ist es, *plausible* Modelle zu konstituieren. Dies erfolgt streng nutzergetrieben, auf eine automatische Generierung von Entwurfsobjekten oder deren Parameterpropagation wird in den genannten Tools verzichtet. Sie sind damit in die Kategorie der 'Design Repositories' einzuordnen, da sie lediglich Klassen von Entwurfselementen des jeweiligen Entwurfsaspekts bieten. Entwerfen bedeutet hier Instanziieren von Entwurfselementklassen sowie Modifizieren und Arrangieren dieser Instanzen.

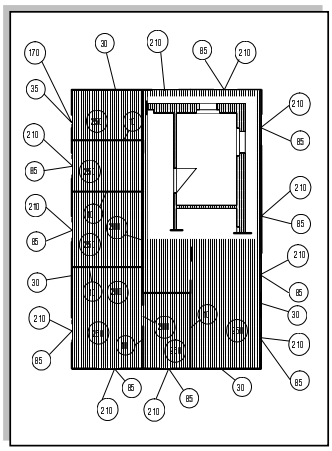
Als Basistools zur Unterstützung des konzeptuellen Entwurfs sind konzipiert:



FUNPLAN realisiert die funktionale Sicht auf das zu erstellende Modell. Es arbeitet über symbolischen Objekten, die Nutzungs- bzw. Struktureinheiten nachbilden. Konstruktive Elemente werden nicht betrachtet. Ergebnis ist eine Spezifikation, die gewöhnlich als Raumprogramm bezeichnet wird (DIN 276/ 277). Das Bauwerksmodell wird initialisiert, indem Objekte erzeugt werden, die den Kernbereich der Aggregation bilden. Der Charakter dieser Objekte ist gemäß der unterstützten Entwurfsphase noch abstrakt und in hohem Maße unscharf.



GENPLAN unterstützt die Formfindung. Es ist geplant, hierzu eine Grundmenge von architektonischen Grundformen der unterstützten Domäne zu bieten (generische 2D- oder 3D-Elemente wie Polygon, Kubus, Pyramide etc. oder spezifische Formelemente der unterstützten Domäne, wie Dächer, Säulen usw.) sowie zugehörige Operationen zum Erzeugen und Editieren des geometrischen Modells (siehe DAHLENBURG [DAH96]). Diese Operationen sind Subtypen der entsprechenden Syntheseoperationen. Insbesondere zur Layoutplanung bieten sich Methoden an, die über flächigen Rasterelemente arbeiten, da diese im Gegensatz zu linienbasierten Skizzentechniken mit raumgreifenden Modellelementen assoziierbar sind. Dies lehnt sich an die bei Architekten gebräuchliche Verwendung von Rasterpapieren in frühen Entwurfsphasen an. Durch die Nutzung von 'VR'-Techniken läßt sich diese Technik auch auf den 3D-Geometrieentwurf (siehe Begriff des 'VOXEL' bei LYSEK, [LYS94])



RELPLAN unterstützt das Fixieren einer Prinzipkonstruktion, in dem konstruktive Entwurfsparameter *spezifiziert* werden. Bei den Entwurfsobjekten handelt es sich wiederum um unscharfe, symbolische Konstruktionselemente. Es sind je nach angestrebten Unterstützungsgrad verschiedene grafische Visualisierungen denkbar (siehe Projektstudien GEBIS [DON96]). Dies gilt auch für bauphysikalische (Ziel-) Entwurfsgrößen, die mit konstuktiven Elementen assoziiert sind, wie etwa Dämmwerte (siehe ECOPLAN [BRA96]).

Abb. 7-1 a-c Tools des PREPLAN-Systems

Die Kopplung der einzelnen Tools zum System ist ‘lose’, alle Tools sind einzeln benutzbar und besitzen eine eigenständige Modellverwaltung, die jeweils demselben Modellierungsparadigma genügt. Der Bezug zwischen den Objekten der verschiedenen Sichten wird durch ausgezeichnete Relationen realisiert, wie sie etwa VANNEDERVEEN beschreibt ([NED93], Abb. 7-2). Auf diesen Relationen arbeiten später Evaluierungsprozesse, wie die Verfahren zur Konsistenzsicherung zwischen Modellsichten (Entwurfsaspekten).

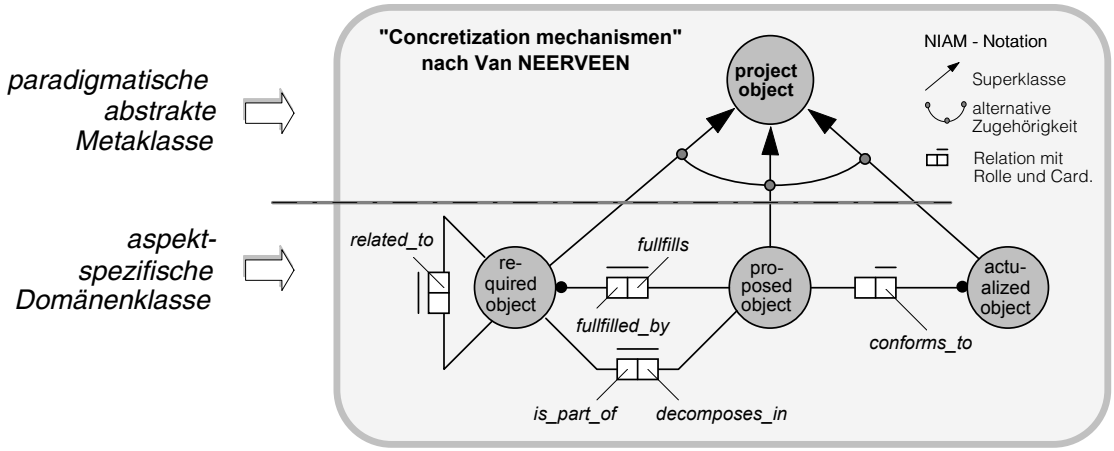
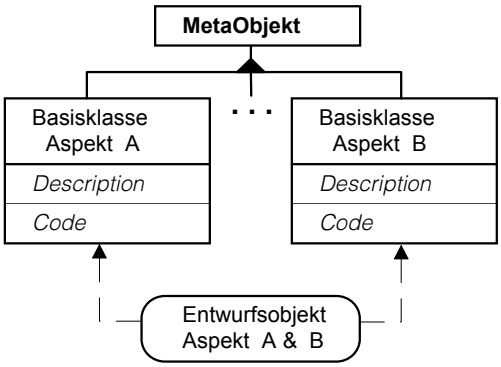


Abb. 7-2 Integration von Objekten verschiedener Entwurfsaspekte durch einheitliche Metaobjektklasse (nach [NED93])

Eine zentrale Bedeutung hat bei diesem Vorgehen die abstrakteste verfügbare Klasse, bei VANNEDERVEEN das ‘*project object*’. Diese Klasse trägt keinerlei domänen- oder aspektspezifische Eigenschaften. Sie bildet die Wurzelklasse der Taxonomie und stellt nur Merkmale, wie die beschriebenen paradigmatischen Relationen bereit bzw. Dienste, wie Zugriffs- und Speichermethoden, um auf dem Wege der Erbschaft für alle Objekte des Modells ein einheitliches Interface zu gewährleisten. Diese ‘MetaObjekt’-Klasse ist Teil des Metamodells und ist als solche unter jedem aspektspezifischen Objektsystem verfügbar (Abb. 7-2).
Da die Zugriffsmethoden Teil des zentral implementierten MetaObjekts sind, hat jedes Tool, das sich von diesem MetaObjekt ableitet, Zugriff auf alle Informationen des jeweils anderen Aspektes. Das MetaObjekt selbst realisiert also die einheitliche Schnittstelle zwischen den Tools. Die technischen Möglichkeiten zur Implementation der Schnittstelle sind vielfältig und reichen von Files, über Prozeßkommunikation im genutzten Betriebssystem, Datenbanktechniken bis hin zu betriebssystemneutralen Objektinterfaces wie CORBA[1].

Enge Kopplung *(Integration)*

Ein einheitliches Objektsystem,
keine Trennung von Objektdeskription und Objektfunktionalität,
strenge Objektidentität,
wechselseitige Kenntnis der Modellaspekte nötig



[1] Common Request Broker Architecture, standardisiertes Objectinterface [COR92]

Lose Kopplung

Ein einheitliches Metamodell zur Modellstrukturierung, eigenständige Domänenmodelle, keine wechselseitige Kenntnis der Domänenmodelle notwendig, Trennung von Objektdeskription und Objektfunktionalität

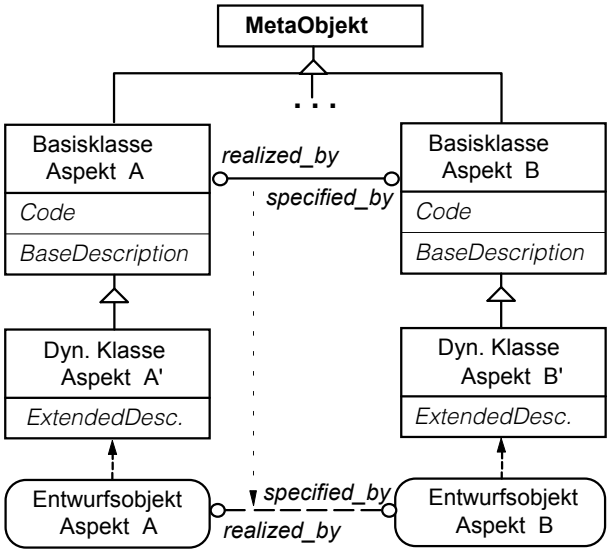


Abb. 7-3 a, b Konzepte zur Kopplung von Modulen zum System

Das Erzeugen dieser Relationen zwischen zusammengehörigen Instanzen verschiedener Modellsichten erfolgt in PREPLAN manuell. Ist dies geschehen, kann ein Evaluierungsprozeß feststellen, ob Formelemente ihrer funktionalen Spezifikation genügen, ob Konstruktionselemente eine geplante Form realisieren, ob konstruktive Elemente des Detailentwurfs die spezifizierte Prinzipkonstruktion verwirklichen usw. (Abb. 7-4).

Zwischen den Tools von PREPLAN werden Objekte einander über die Phasengrenzen hinweg durch eine 1:1 Relation zugewiesen, d.h. zu einen Funktionselement gehört genau ein Realisierungselement. Die Prüfung gleichnamiger Parameter ist hier ein trivialer Prozeß, was im Rahmen eines 'design repository' akzeptabel ist. Für einen 'design assistant' kann die Relation 'realized_by' aber von der Kardinalität n:m sein, entsprechende Evaluierungsfunktionen wären dann wesentlich komplexer und von sehr spezifischer Semantik.

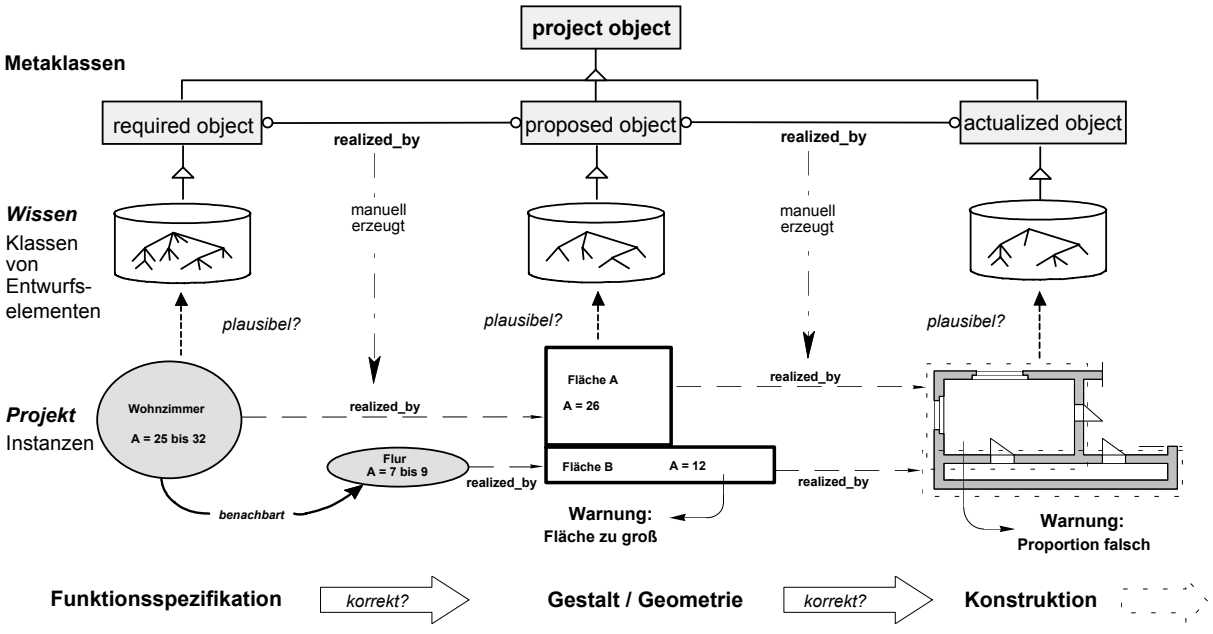
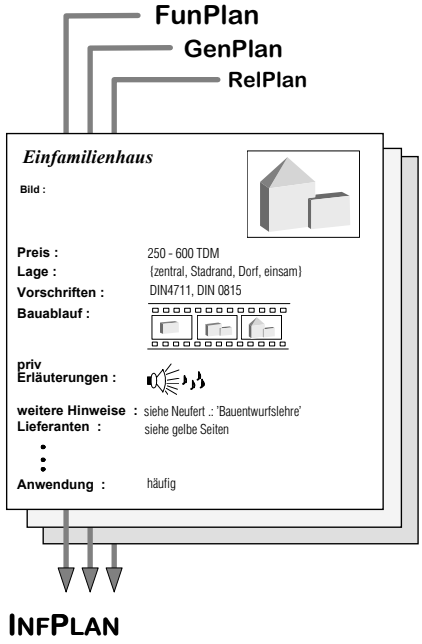


Abb. 7-4 'Lose' Toolkopplung durch manuell applizierte Relation *'realized_by'*

Wie bereits ausgeführt, ist eine vollständige Formalisierung aller projekt- bzw. domänenrelevanten Informationen weder möglich, noch sinnvoll. Die Tools des PREPLAN-Konzepts gehen von einer reduzierten Berechenbarkeit im Bauwerksmodell für frühe Entwurfsphasen aus. Besonders in den von PREPLAN unterstützten Entwurfsphasen werden Informationen durch den Architekten nur schwach strukturiert. Soweit sie überhaupt mit dem Computer erfaßt werden können, liegen sie in semantikfreien, neutralen Dokumentenformaten vor. Hierzu zählen Textdokumente, Bilder (Fotos) etwa zur Dokumentation baulicher Situationen, akustische Kommentare, aber auch neutrale oder vom vorliegenden Metamodell abweichende CAD-Modelle (z.B. AutoCAD Zeichnungen). Für ein CAAD-Systemkonzept stellen somit Tools zur Verwahrung nichtformalisierter Informationen eine sinnvolle Erweiterung dar. Da die Verwaltung solcher multimedialer Dokumente und deren Integration in das aktuelle Domänenmodell für jedes Tool eines Systems geleistet werden muß, ist hierfür das Tool INFPLAN zur umfassenden Nutzung vorgesehen. Es dient dem Retrieval und Authoring entsprechender multimedialer Modellerweiterungen. Da die Funktionalität eines solchen Tools nicht vom unterstützten Entwurfsaspekt abhängt, kann es aus einer Reihe von Einzeltools heraus Verwendung finden. Die Anpassung an den aktuellen Entwurfsaspekt erfolgt beim Laden des jeweils zugehörigen formalen Wissens in Form der Taxonomie mit den dort gespeicherten Verweisen auf das jeweils spezifische informale Wissen.



INFPLAN ist ein phasenneutrales Universaltool zum Retrieval und Authoring nichtformalisierter Informationen in multimedialen Dokumenten. Alle mit einem Entwurfsobjekt assoziierten Dokumente werden in einem '*DescriptionObject*' gekapselt und über eine spezifische Relation in die Beziehungsstruktur der Entwurfsobjekte sowie deren Klassen integriert. Zur inhaltlichen Strukturierung informalen Wissens (bei Klassen) bzw. Projektdaten (bei Instanzen) dienen neben speziellen Links des Hypermediasystems auch die spezifischen, formalisierten CAD-Relationen des betrachteten Entwurfsaspekts.

Abb. 7-5 Tool INFPLAN zur Integration nichtformalisierter Informationen

Der Mechanismus der 'losen Kopplung' sichert, daß informale Teile eines Spezifikationsobjektes ebenfalls in der nächsten Entwurfsphase verfügbar sind. Obwohl eine automatische Evaluierung derartiger nichtformalisierter Informationen nicht möglich ist, läßt sich mit ihrer Hilfe die Erfüllung einer Spezifikation gewährleisten, indem eine manuelle Kontrolle durch den Nutzer angefordert wird. Er muß dann mit expliziter Bestätigung die Korrektheit quittieren. Auf diesem Weg kann INFPLAN zur Systematisierung des Entwurfsprozesses beitragen.

Von den angeführten Tools sind gegenwärtig FUNPLAN und INFPLAN prototypisch implementiert. Die Begründung für diese Auswahl ergibt sich daraus, daß gegenwärtig

- kaum technische und methodische Erfahrungen mit CAD-gestützten Aufbau von Bauwerksspezifikationen vorliegen
- informale, insbesondere multimediale Annotationen in (berechnungsorientierten) CAD-Systemen nicht unterstützt sind.

Die Gesamtstruktur der gegenwärtig konzipierten Tools im System PREPLAN zeigt Abb. 7-6.

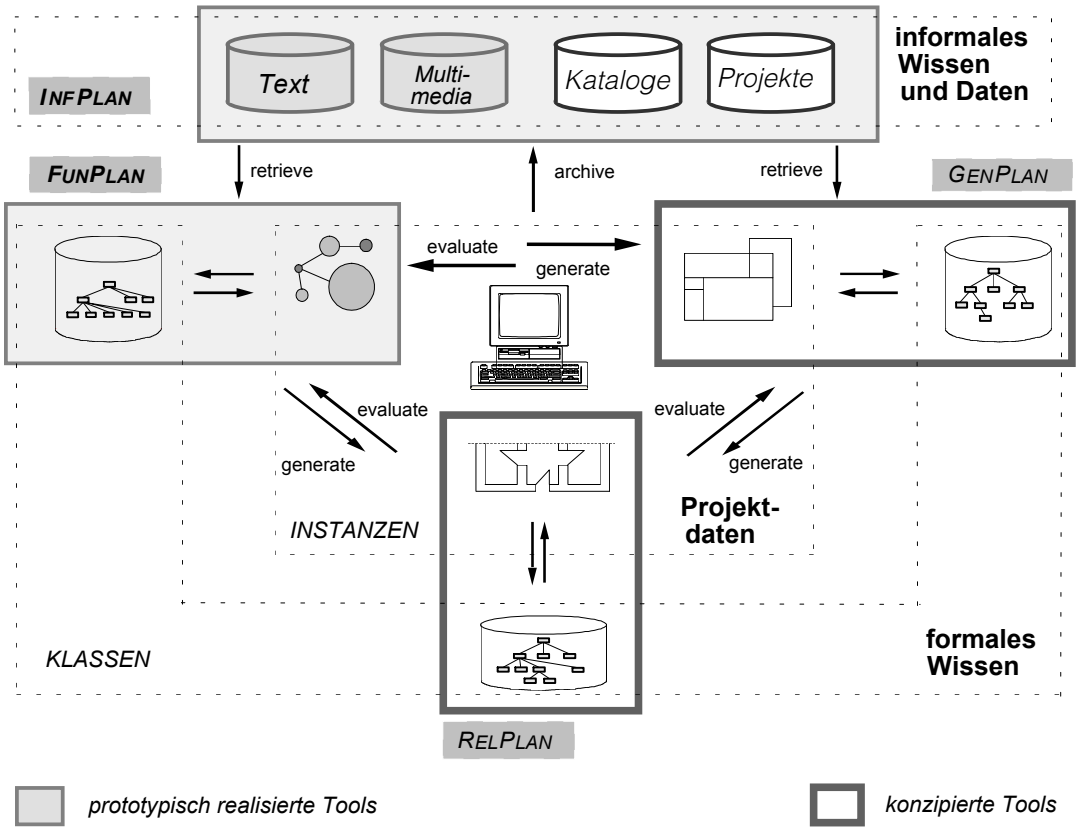


Abb. 7-6 CAAD-Werkzeuge des Systems PREPLAN

7.2 FUNPLAN

Als funktionale Spezifikation von Bauwerken kann ein Graph ('*bubble diagrams*') verstanden werden, dessen Knoten räumlich-strukturellen Einheiten entsprechen. Die Kanten dieses Graphen beschreiben Relationen zwischen Modellobjekten, die gelten bzw. gelten sollen. Knoten und Kanten werden in der planerischen Praxis näher durch textuelle oder grafischen Annotationen beschrieben. Im folgenden wird eine derartige explizite Darstellung für funktionale Spezifikationen zu grunde gelegt.

Der Prototyp von FUNPLAN sollte Aufbau und Strukturierung solcher Graphen gemäß des objektorientierten Modellierparadigmas gestatten. Die bewährte Planungstechnologie auf der Basis grafisch repräsentierter Relationennetzwerke sollte erweitert werden um

- einen Vorrat an abstrakten vorgedachten Planungsobjekten
  (Wissen über Planungsobjekte)

- die Prüfung konkreter Planungsobjekte gegen Wissen über diese Objekte
- eine erweiterte Beschreibbarkeit für Planungsobjekte
- eine freie Änderbarkeit der Plankomplexität durch rekursives Zusammenfassen oder Expandieren von Planungsobjekten in einer hierarchischen Aggregation.

Der Prototyp sollte gestatten, Pläne soweit als möglich grafisch zu erstellen, zu editieren und zu repräsentieren. Gemäß des Charakters von Entwurfsobjekten in der zu unterstützenden Planungsphase müssen unscharfe Beschreibungen möglich sein.

Die gewünschte Funktionalität wurde in Form von zwei Programmodulen realisiert[2]:

FUNPLAN - *Wissen* ⟷ **FUNPLAN - *Projekte***

- Klasseneditor zum Aufbau einer taxonomischen Wissensbasis
- Gewährleisten der Plausibilität, wie sie aus der mengentheoretischen Betrachtung für hierarchische Klassensysteme folgt

- Instanzeditoren für Planungsobjekte und Plausibilitätstest (formalisierte und informale Attribute und Relationen)
- Grafischer Editor für Funktionsplangraphen (Topologie und relevante geometr. Eigenschaften)
- Navigation in komplex aufgebauten Funktionsgraphen
- Einfache numerische Auswertungen über komplexe aufgebaute Planungsobjekte

7.2.1 FUNPLAN - Wissen

Jegliche Art modellhafter Repräsentation bedarf eines Kontextes, in dem sie interpretiert wird. In der Objektorientierung bildet Wissen, das in Form des Klassensystems abgebildet wird, diesen interpretatorischen Rahmen. Sowohl für die rechentechnische Umsetzung als auch für die planerische Praxis bietet das OO-Modellierungsparadigma den Vorteil, daß aktuelle Planungsobjekte in Form von Instanzen aus der Wissensbasis (von Klassen) abgeleitet werden. Instanzen verfügen per Definition über dieselbe Struktur und über dasselbe Verhalten wie die Klassen als Wissenselemente.

FUNPLAN unterstützt folgende Arten von Metamodellelementen für formales Wissen:

- KLASSENOBJEKTE

Klassenobjekte können zur Laufzeit durch den Nutzer erzeugt, umbenannt, angezeigt und gelöscht werden. Dies erfolgt durch grafisches Selektieren in einem Klassenbrowser, der ein entsprechendes Menü anzeigt. Klassen sind in einen hierarchischen Klassenbaum geordnet, alle Klassenmerkmale werden einfach vererbt. Die dynamische Änderbarkeit des Wissens durch den Nutzer gestattet eine fortwährende Modellmigration, um sie dem aktuellen Stand des Planungswissens anzupassen. Da diese Änderungen durch Nutzer erfolgen, können spezifische, personalisierte Domänenmodelle erstellt werden. Da (gegenwärtig) der erforderliche Kenntnisstand bei den Nutzern nicht vorausgesetzt werden kann, dürfte in der Praxis dieser Weg der Personalisierung eher von CAD-Herstellern zur spezifischen Konfiguration ihrer Systeme beschritten werden.[3]

[2] siehe Anhang C
[3] Der Facility Managemen Systemkern KOPERNIKUS (Hochtief Software) beschreitet diesen Weg, stellt die Möglichkeiten zur Dynamisierung jedoch nicht dem Nutzer zur Verfügung.

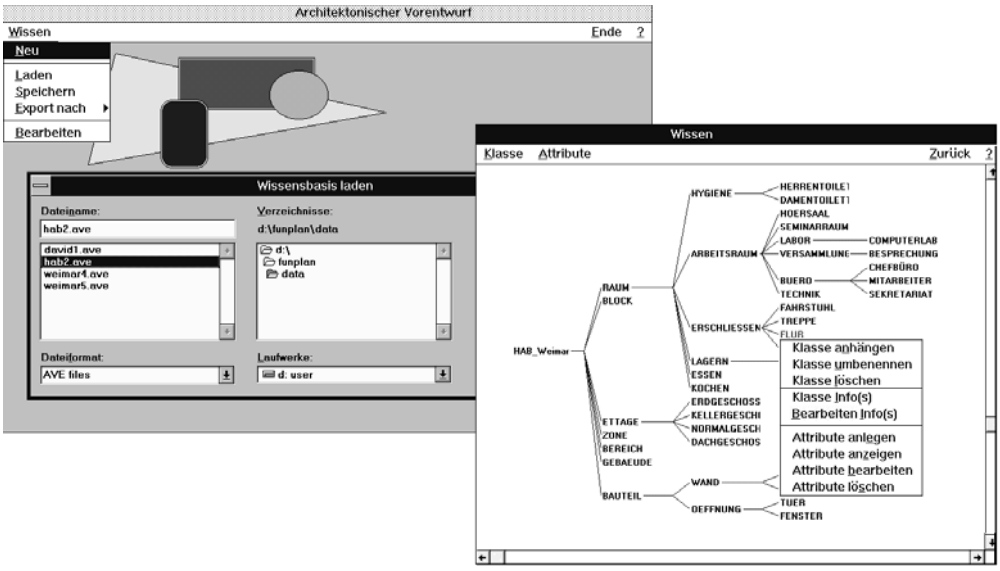


Abb. 7-7 Klassenbrowser und Klassenmenü

- ATTRIBUTE VON OBJEKTEN

Für die formalisierte Beschreibung von Objekten wird im Prototyp lediglich eine Grundmenge von Attributtypen angeboten, die sich aus den Möglichkeiten des Implementationstools KappaPC ergab. Dies deckt zwar weitgehend den Bedarf ab, wie er aus der Entwurfspraxis resultiert, jedoch nicht die Beschreibungsmächtigkeit, die durch Nutzung des OO-Modellierungsparadigmas auf einem Computer möglich wird. Als reflexive Attributtypen werden lediglich numerische Attribute sowie listenwertige symbolische Attribute angeboten. Über das Tool INFPLAN sind beliebig mediale Annotationen für Klassen erzeugbar. Attribute können zur Laufzeit durch den Nutzer an jeder Stelle des Klassenbaums erzeugt, umbenannt, angezeigt und gelöscht werden. Sie werden dynamisch an Subklassen vererbt (dies gilt auch für deren Löschung).
Als fixe, durch den Nutzer nicht änderbare Attribute sind im Wurzelobjekt der Taxonomie implementiert :

– Fläche, Position (x, y), Höhe, Proportion (dx/dy), alle vom Typ numerisch
– Orientierung im Grundriß (‘`Nord`’, ‘`Nordost`’, ... , ‘`Zentral`’), Typ symbolisch

- RELATIONEN ZWISCHEN OBJEKTEN

Relationen als transitive Merkmale von Objekten sind als relationaler Attributtyp implementiert. Strukturell handelt es sich um unbewertete 1:n Relationen.
Als fixe, durch den Nutzer nicht änderbare Relationen sind im Wurzelobjekt der Taxonomie implementiert: ‘*Exist*’, ‘*Not_Exist*’. Diese Relationen sind nur auf Klassenebene verfügbar. Sie spezifizieren die gewünschte Existenz bzw. Nichtexistenz eines Planungsobjektes einer bestimmten Klasse in Abhängigkeit von der Existenz eines Objektes einer anderen Klasse. Die Aggregationsrelation ist im gegenwärtigen Prototyp dagegen auf Klassenebene nicht verfügbar. Das bedingt, daß Standardlösungen aus komplex aufgebauten Planungsobjekten, wie etwa in Systemen, die sich eines Skelettkonstruktionsverfahrens bedienen ([PUP91]), nicht verfügbar sind. Wegen des Unikatcharakters von Bauwerken ist das aber auch nicht wünschenswert.

- STANDARDANNAHMEN FÜR MERKMALE UND RELATIONEN ZUR INITIALEN BELEGUNG

Für alle Attribute und Relationen können Standardwertbelegungen vorgesehen werden. Diese Initialisierung ist allerdings optional, im System muß deshalb ein expliziter Wert für unbekannte Wertebelegung verfügbar sein ('UNKNOWN'). Für Relationen verweist der Standardwert bei Klassenobjekten auf andere Klassenobjekte. Die Semantik der Vorbelegungen für Relationen bei den Klassen sagt aus, daß Instanzobjekte dieser Klassen dann standardmäßig über die entsprechende Relation verfügen sollen, wenn Planungsobjekten aller an der Relation beteiligter Klassen existieren. Alle Wertebelegungen in Attributen und Relationen unterliegen ebenfalls der Vererbung (*Wert*vererbung).

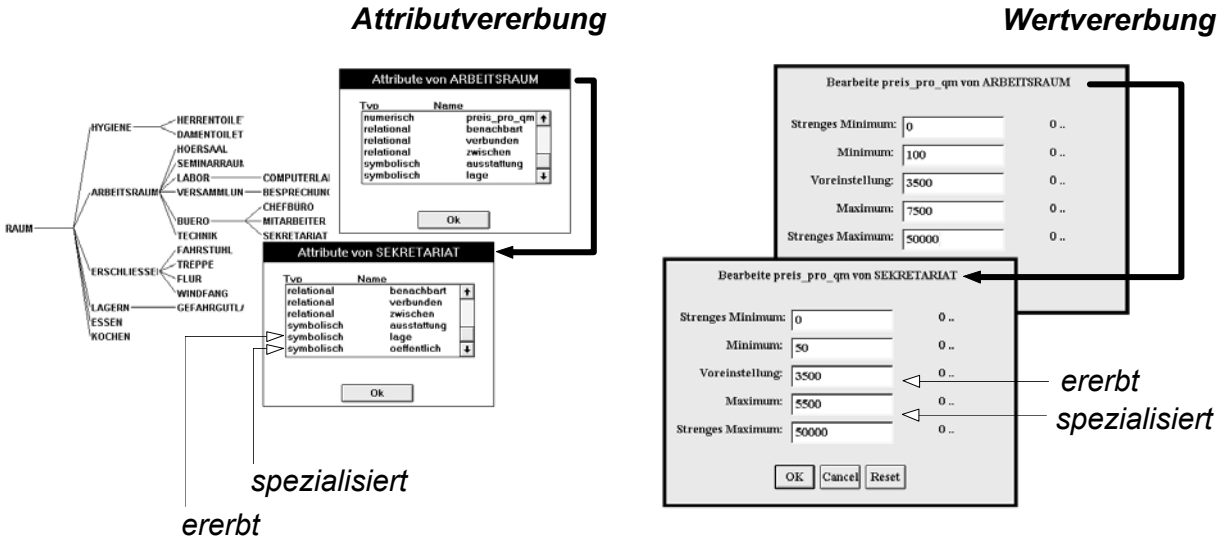


Abb. 7-8 Typen von Attributen und deren Vererbung

- RANDBEDINGUNGEN FÜR ATTRIBUTE UND RELATIONEN

Die Objektorientierung nutzt heute hauptsächlich die Einschränkung/Erweiterung von Merkmalsmengen von Klassen entlang der Generalisierungsrelation in hierarchischen Klassensystemen, wie sie aus der Betrachtung entsprechender Mengensysteme folgt. Die Mengenrelationen gelten jedoch auch für die Wertebelegung der Merkmale selbst. Die ererbten Werte können lokal überschrieben werden. Um die Plausibilität eines Werteeintrags zu sichern, wurde ein System von Hard- und Softconstraints für die implementierten Merkmalstypen entwickelt und in Facetten implementiert (Abb. 7-8 und Tab. 7-1). Auf Grund der Facettisierung von Werten in FUNPLAN mußte die Wertevererbung entlang der Subklasse- und Instanzrelationen nachimplementiert werden. Der Wert der initialen (Standard-) Belegung ist hier, im Gegensatz zur gebräuchlichen OO-Sicht, ebenfalls eine Facette.

Die angegebenen Facetten spezifizieren Mengen zulässiger Wertebelegungen in den Attributen bzw. Relationen. Entlang der Spezialisierungsrelation gilt für eine vererbte Wertemenge $WM_{i,sub}$ bei einer Subklasse K_{sub} initial $WM_{i,sub} = WM_{i,super}$, mit $WM_{i,super}$ mögliche Belegungen eines Merkmals M_i bei einer Superklasse K_{super}.

Bei der Veränderung dieser ererbten Wertemenge ist zu sichern, daß im Sinne der Mengenrelation $K_{super} \supseteq K_{sub}$ gilt $WM_{i,super} \supseteq WM_{i,sub}$, denn nur dann ist das Wissen in der gegebenen Modellstrukturierung plausibel (siehe Abb. 7-10). Dies gilt analog für die Instanzrelation.

Attributtyp	Facette	Test	Semantik
numerisch	Standardwert x	–	Standardannahme für den Attributwert, solange er nicht überschrieben wird
☑	Strenges Maximum $smax$	$x \leq smax$	Verbot, Werte größer als *smax* werden nicht akzeptiert
✓	Maximum max	$x \leq max$, $max \leq smax$	Warnung, Werte größer *max* werden erst nach Bestätigung akzeptiert
✓	Minimum min	$min \geq smin$, $x \geq min$	Warnung, Werte kleiner min werden erst nach Bestätigung akzeptiert
☑	Strenges Minimum $smin$	$x \geq smin$	Verbot, Werte kleiner als *smin* werden nicht akzeptiert
symbolisch relational	Standardwert X	–	Standardannahme für die Attribut- bzw. Relationenwerte, solange er nicht überschrieben wird
☑	Vorschrift Vo	$X \subseteq Vo$	Verbot, wenn die neuen Werte nicht vollständig in *Vo* enthalten sind, werden sie nicht akzeptiert
✓	Empfehlung E	$X \subseteq E$, $E \supseteq Vo$	Warnung, wenn die neuen Werte nicht vollständig in *E* enthalten sind, werden sie erst nach Bestätigung akzeptiert
✓	Warnung W	$W \cap Ve = \emptyset$, $X \cap W = \emptyset$	Warnung, wenn ein neuer Wert Mitglied von *W* ist, wird er erst nach Bestätigung akzeptiert
☑	Verbot Ve	$X \cap Ve = \emptyset$	Verbot, wenn ein neuer Wert Mitglied von *Ve* ist, wird er nicht akzeptiert

✓ – Softconstraint (kann verletzt werden) ☑ – Hardconstraint (darf nicht verletzt werden)

Tab. 7-1 Facetten in FUNPLAN und deren Semantik

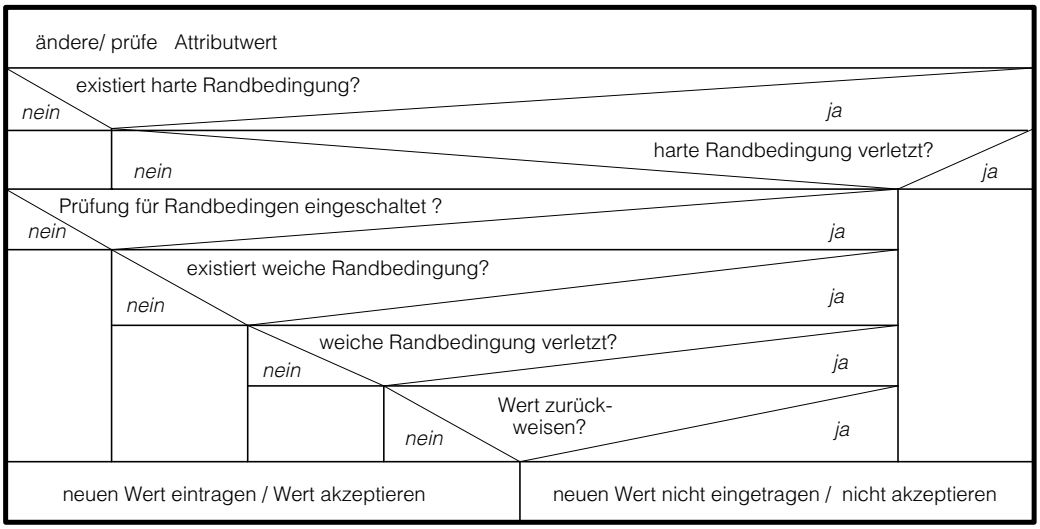


Abb. 7-9 Algorithmus zur Constraintauswertung

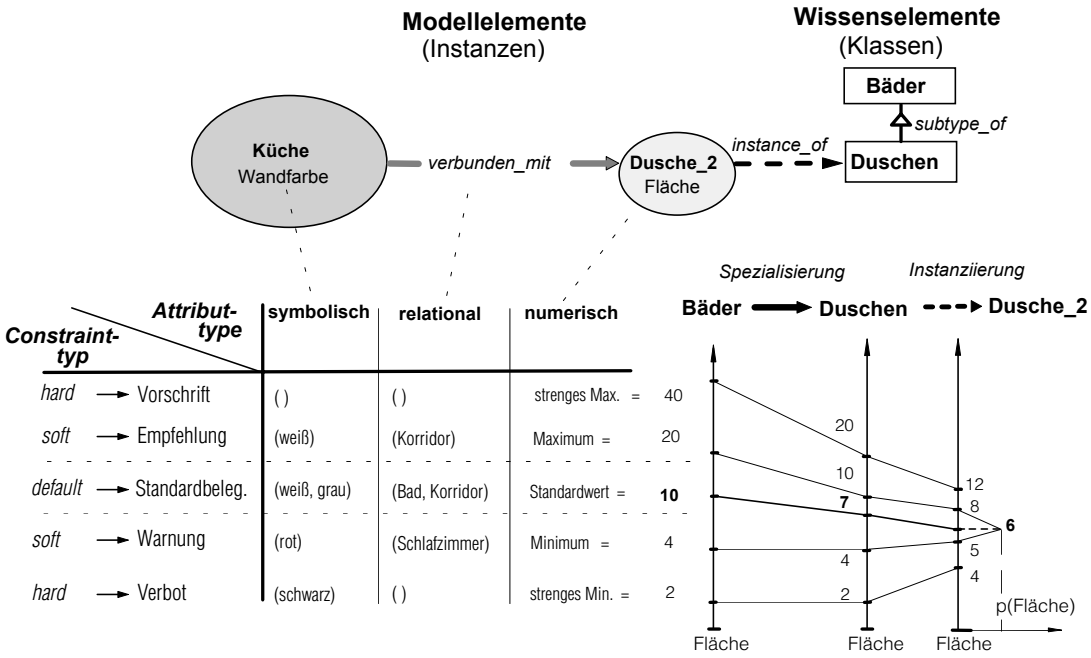


Abb. 7-10 Attributwertvererbung

Da Instanzen als Konkretisierungen von Klassen gebildet werden, können die Verfahren zur Beschreibung von Klassenobjekten sowie zur Vererbung und Plausibilitätssicherung zwischen Klassenobjekten, homogen auch auf Instanzobjekte angewandt werden. In Applikationen können Klassen und Instanzen äquivalent genutzt werden (etwa zur Abbildung relationaler Unschärfe) sowie technisch ähnlich implementiert werden.

7.2.2 FUNPLAN - Projekt

Der Projektmodul stellt im Verbund mit FUNPLAN-Wissen das eigentliche Planungswerkzeug dar. In ihm wird die OO-Modellierungsmethodik zum Erstellen grafischer Funktionspläne genutzt.

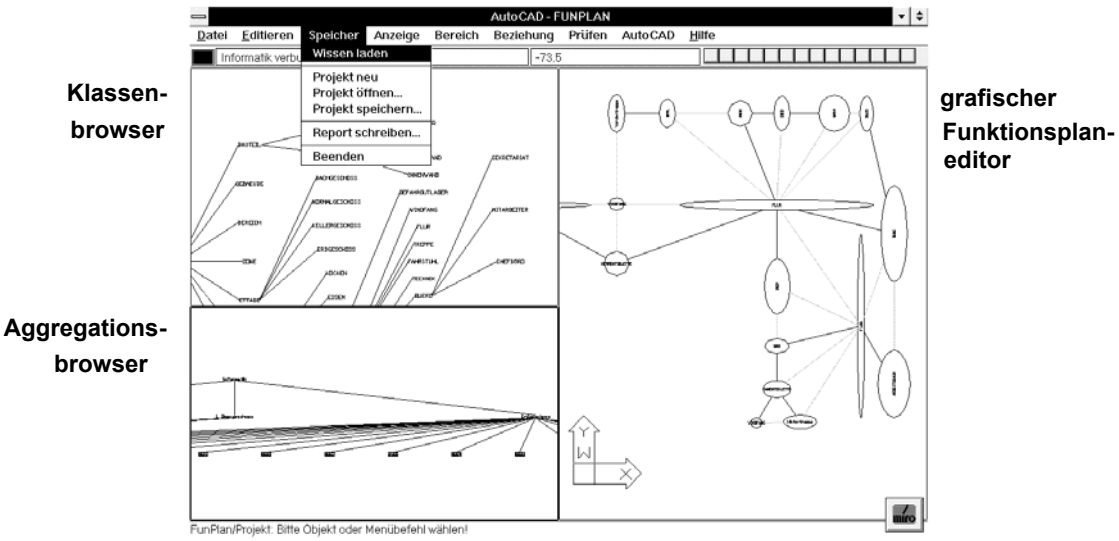


Abb. 7-11 Fensterteilung FUNPLAN-Projekt

Die grafische Repräsentation der ‘bubble diagramms’ wird der visuell orientierten Arbeitsweise des Architekten eher gerecht als deren Abbildung in einer Adjazenzmatrix, obwohl auch diese alphanumerische Darstellung funktionaler Bauwerksspezifikationen genutzt wird. Der Programmodul FUNPLAN-Projekt bietet unter dem Implementationstool AutoCAD drei starre Fenster (Abb. 7-11), die ein grafisch-interaktives Erstellen von Funktionsplänen gestatten.

Zur funktionalen Spezifikation und Projektverwaltung bietet FUNPLAN folgende Funktionalität:

- KLASSIFIKATION BZW. REKLASSIFIKATION VON OBJEKTEN

Planungsobjekte werden immer als Instanzen von Klassen gebildet. Die Klasse wird im Klassenbrowser selektiert. Der Prozeß der Klassifikation ist dynamisch, d.h. Instanzen lassen sich beliebig spezialisieren, abstrahieren oder umklassifizieren. Dies ermöglicht einerseits die Abbildung von Abstraktionsunschärfe sowie andererseits die dynamische Neuinterpretation eines Planungsobjektes in zeitlich oder individuell differierenden Klassenbeschreibungen (Revision).

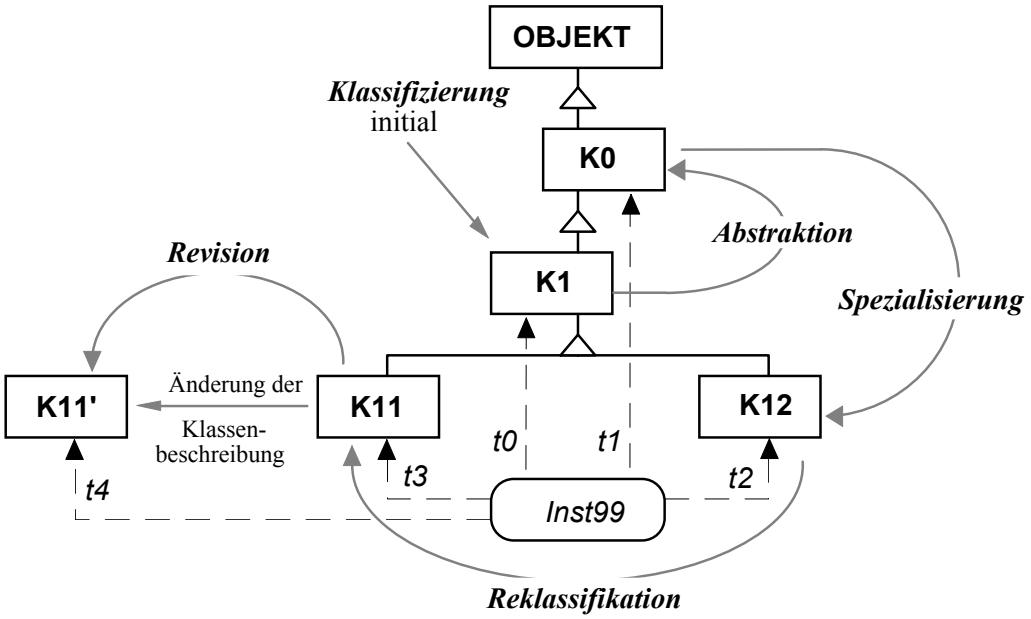


Abb. 7-12 Dynamische Klassifiktion in FUNPLAN

Bei der Neuinterpretation eines Planungsobjektes ist als stärkste Form der Abstraktion die Klasse ‘OBJEKT’ immer verfügbar. Sie entspricht dem ‘MetaObjekt’ des Metamodells, die als solche nicht durch den Nutzer geändert werden kann. Ein Problem bei der dynamischen Interpretation/Klassifikation von Planungsobjekten ist, wie man mit Informationen verfährt, die bei der Ursprungsklasse K_{quelle} formalisiert waren, in der Zielklasse K_{ziel} jedoch nicht. Im Beispiel aus Abbildung 7-13 betrifft das die Attribute ‘p’ und die Relation ‘c’.
Im Sinne des angestrebten ‘design repository’ werden diese als nichtformalisierte Informationen in einer geeigneten Strukturierung im System erhalten. Wichtigste Grundvoraussetzung hierzu ist, daß Instanzen den gesamten Erbschaftspfad bis hin zur Klasse ‘OBJEKT’ kennen (!) und ggf. persistent speichern.

So ist garantiert, daß immer eine gemeinsame Oberklasse K_{trans} gefunden wird. Im Sinne der Mengen gilt für diese Klasse K_{trans}, daß

$$\left(K_{quelle} \subseteq K_{trans}\right) \wedge \left(K_{ziel} \subseteq K_{trans}\right)$$

und

$$\neg\exists K_{sub} \;\middle|\; \left((K_{sub} \subseteq K) \wedge (K_{ziel} \subseteq K_{sub}) \wedge (K_{quelle} \subseteq K_{sub})\right).$$

Hieraus folgt für die Menge der Merkmale $MInst_{trans}$, die für eine Instanz bei solch einer Neuklassifizierung von K_{quelle} (Merkmalsmenge MK_{quelle}) nach K_{ziel} (MK_{ziel}) übertragen werden kann $MInst_{trans} := MK_{quelle} \cap MK_{ziel}$. Durch die Definition des hierarchischen

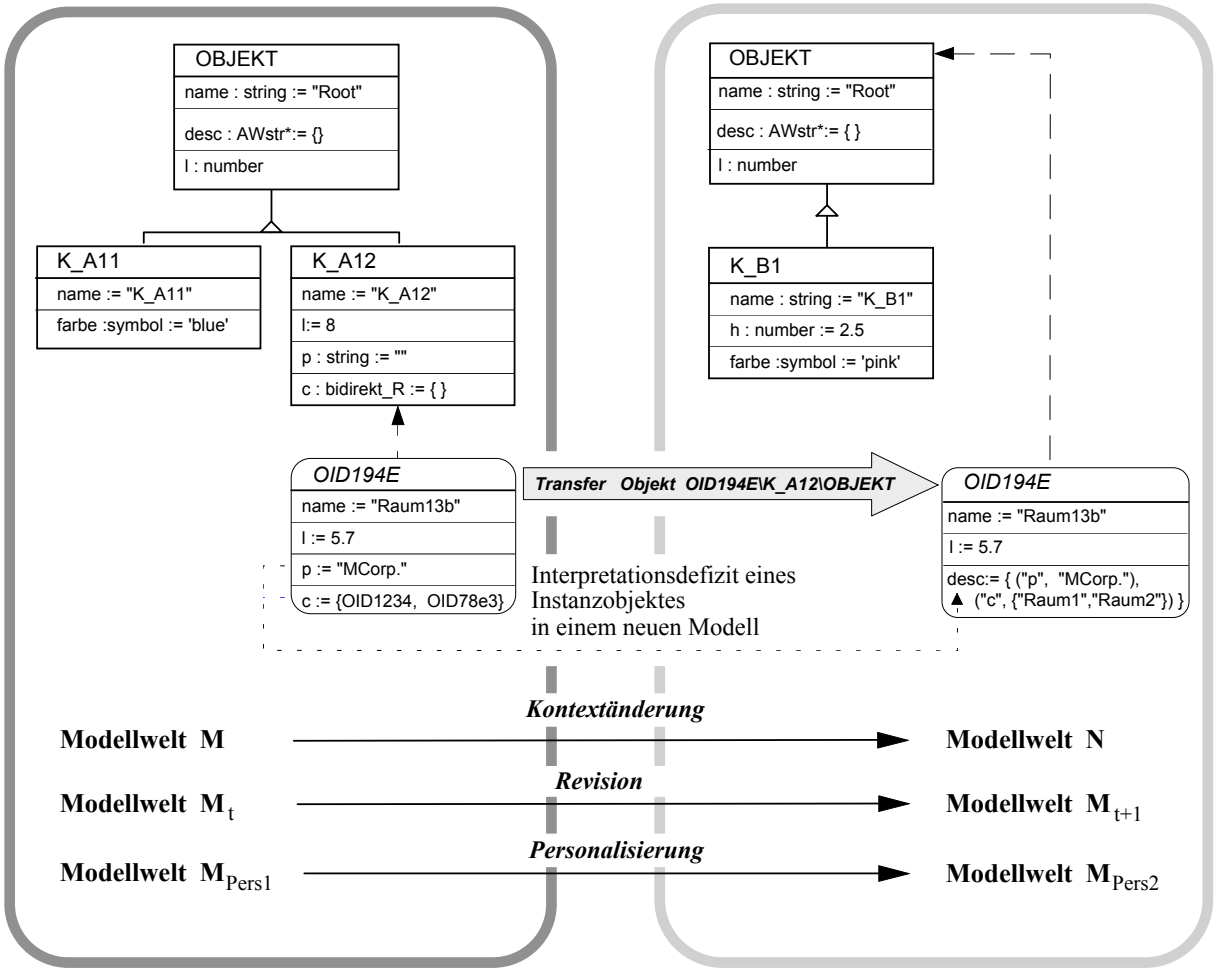


Abb. 7-13 Dynamische Klassifikation als Interpretationsprozeß

Mengensystems folgt, daß mindesten die Menge an Merkmalen MK_{Root} übertragen werden kann, die der Wurzelklasse K_{Root} des Klassensystems zugeordnet ist. Im vorliegenden Anwendungsfall sind das die Attribute und Relationen, die für die Applikation fest in 'OBJEKT' definiert wurden.

Die Menge der formalen Merkmale M_{diff}, die bei der Umklassifizierung verlorengehen, ist $M_{diff} := MK_{quelle} \setminus MK_{trans}$. Neu verfügbar und über die Vererbung mit Standardwerten belegt wird die Merkmalsmenge M_{add}, für die gilt $M_{add} := MK_{ziel} \setminus MK_{trans}$. Sinnvoll ist es, diese Merkmalsmenge als informale Beschreibung (etwa in Form geeignet strukturierter Strings) bei der Instanz zu erhalten (siehe Abb. 7-13 und 7-14).

Für die Planungsobjekte (Instanzen) lassen sich Strings als Kommentierung des Objekts angeben. Sie sind als 'Kurztext' in der Instanzbeschreibungsmaske verfügbar und dienen gegenwärtig auch zur Aufnahme informaler Merkmals/Wert-Paare, die bei einer dynamischen Klassifizierung entstehen können.

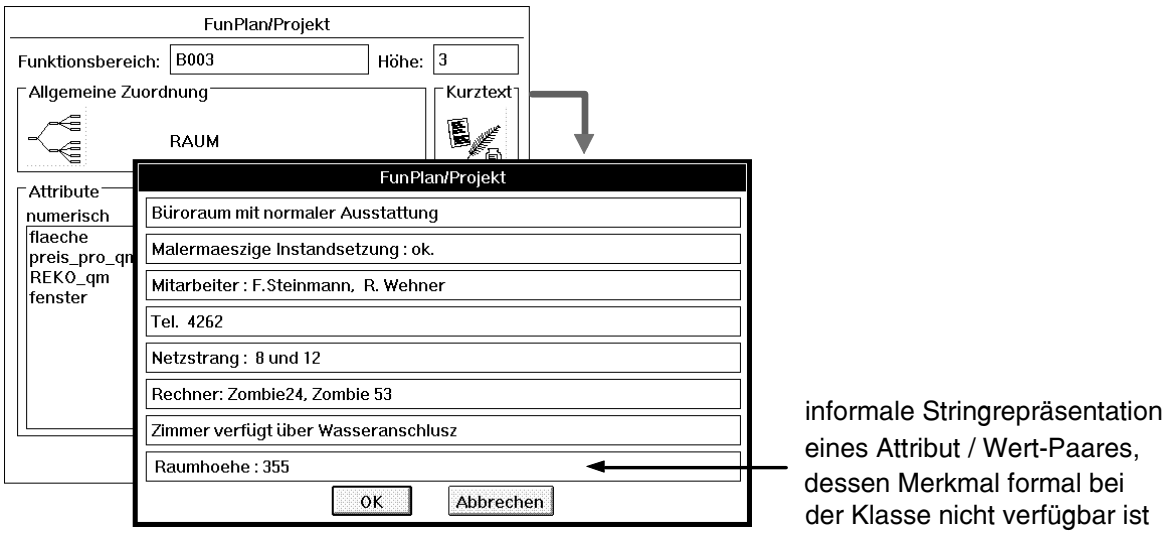


Abb. 7-14 Kurztexte als informale Beschreibung in FUNPLAN-Projekt

- MERKMALE VON PLANUNGSOBJEKTEN

Die Funktionalität zur Beschreibung von Planungsobjekten entspricht weitgehend der des Klasseneditors. In einem ersten Prototyp des Systems wurden tatsächlich dessen Funktionen genutzt. Da die hierfür verwendete DDE-Kopplung[4] zu inakzeptablen Laufzeiten führte, wurden die entsprechenden Funktionen unter AutoCAD neu erstellt. Die wenigen in FUNPLAN-Wissen statisch implementierten Attribute dienen zur Übertragung der formalen Topologie des Funktionsgraphen in einen geometrischen Plan. Knotenobjekte werden flächen- und proportionsrichtig gezeichnet und verfügen über eine Position im 3D-Raum (Abb. 7-15). Alle geometrisch relevanten Attributinformationen können grafisch im Plan editiert werden. Dies gilt auch für die Relationen.

[4] Dynamic Data Exchange, MS Windows Technologie zur Prozeßkommunikation

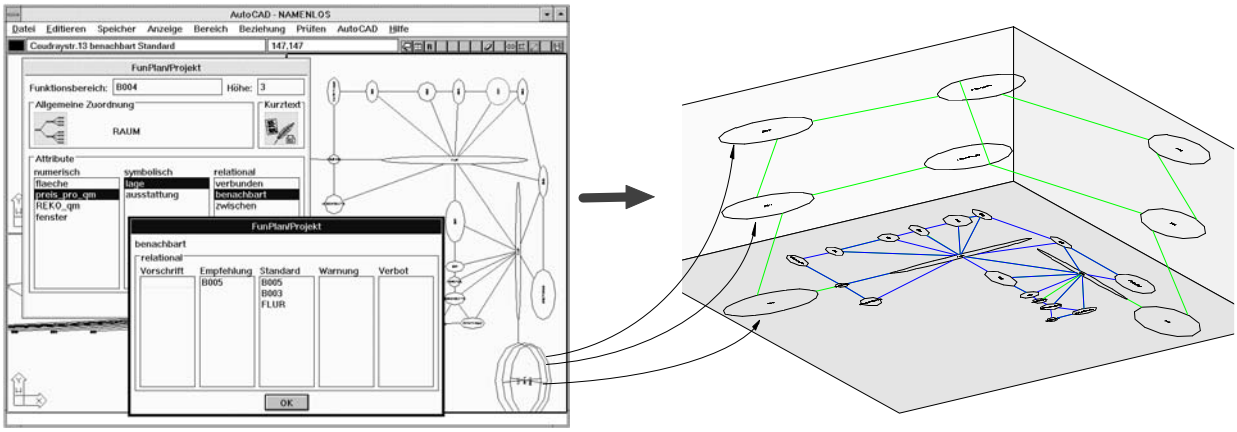


Abb. 7-15 Editmaske für symbolische Attribute und 3D-Funktionsgraph

- RANDBEDINGUNGEN FÜR MERKMALE UND RELATIONEN

Ein wesentliches Ergebnis der Arbeit mit FUNPLAN ist die Erkenntnis, daß in einer Planungsumgebung Unschärfe als Constrainttyp verstanden und implementiert werden kann. Da jeder Plan eine modellhafte Vorwegnahme eines Artefaktes ist, spezifiziert er das Produkt ‘wie es sein soll’. Erfolgt diese Spezifikation unscharf, bedeutet das letztlich eine Relaxation der entsprechenden Constraints.
So wird in FUNPLAN die Facettenstruktur der Merkmale der Wissensebene genutzt, um auf Planniveau attributive Unschärfe von Objekten auszudrücken. Im Planungsprozeß nachfolgende Tools wie etwa GENPLAN, können diese Unschärfe wiederum als Constraints interpretieren.

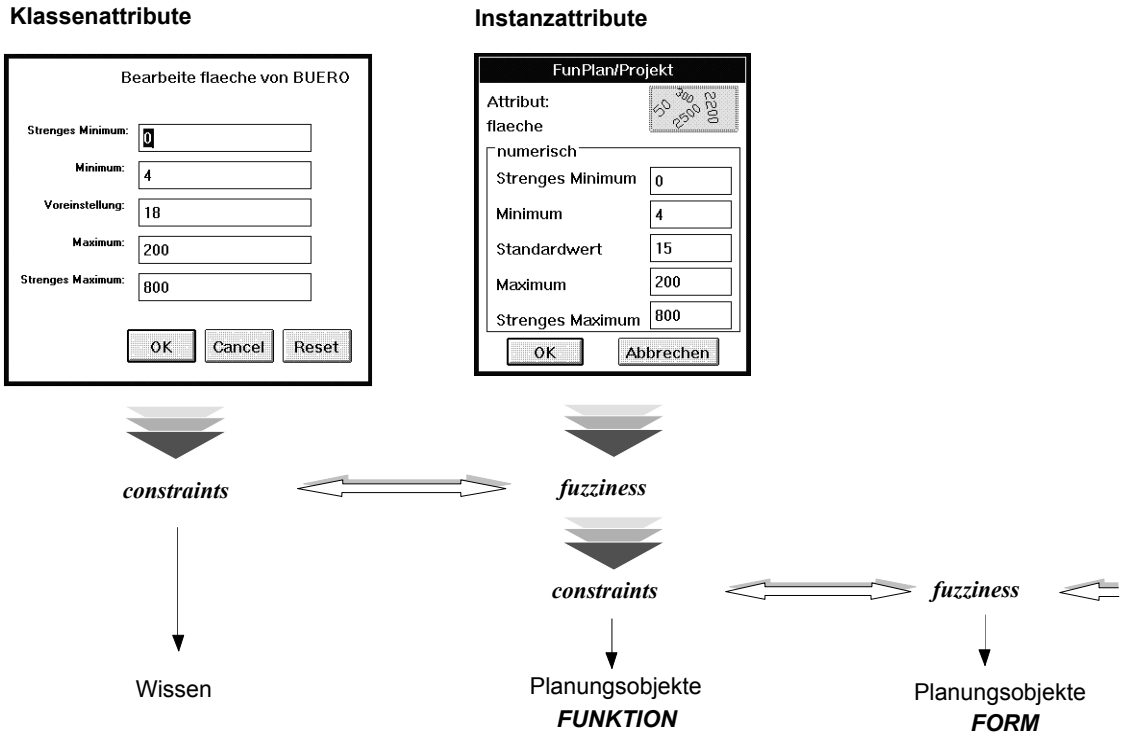


Abb. 7-16 Dualität von Unschärfe und Constraints

Die Benutzung von FUNPLAN zu Lehrzwecken zeigte, daß eine Wissensbasis mit einem solchen System von Zwangsbedingungen einer längerfristigen Entwicklung bedarf, um zu einer produktiven Erweiterung der Entwurfsumgebung zu werden – zu eng gefaßte Constraints führen zu ständigen Warnungen. Wenn im Gegenteil Randbedingungen zu weit gefaßt sind, kann dies im Falle der Übernahme ererbter Merkmalswerte zu Objektbeschreibungen führen, die den Zweck einer Spezifikation nicht erfüllen. Dies betrifft insbesondere Constraints auf Relationen, da sie strukturelle Aspekte des Bauwerksmodells ausdrücken. Deshalb ist im Prototyp die Prüfung für Softconstraints für alle Merkmale abschaltbar. Bei Bedarf kann das gesamte Modell später auf Plausibilität geprüft werden, bzw. können Wertbelegungen, die aus weichen Randbedinungen resultieren, auf Anforderung propagiert werden (Abb. 7-17). Die Anzeige von Relationen und deren Constraints ist steuerbar.

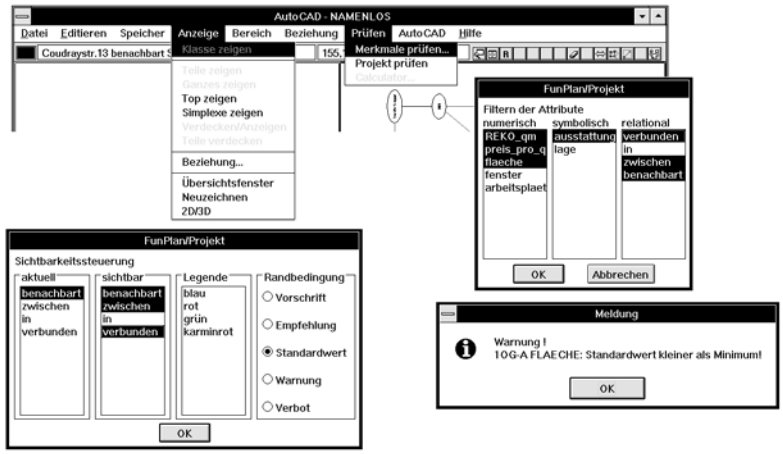


Abb. 7-17 Plausibilitätsprüfung und Relationenanzeige

- AGGREGATION VON OBJEKTEN

FUNPLAN unterstützt hierarchische Aggregationen in frei wählbarer Strukturierungstiefe, um
- eine opportunistische Entwurfsstrategie zu ermöglichen
- Abstraktionen (und damit Unschärfe) auf der Grundlage einer 'Teil-Ganzes'- Relation zu bieten
- einfache numerische Auswertungen zu realisieren.

In FUNPLAN ist der Aggregationsgraph eine Heterarchie mit allerdings genau einem Topkomplex. Dieser Topknoten ist gleichzeitig der Projektname. Der *Focus*[5] der Bearbeitung liegt anfangs auf diesem Topobjekt. Sobald dieses Objekt durch Teile untersetzt wurde, kann der Focus jedoch frei in der Aufbaustruktur verschoben werden (siehe Menü Abb. 7-18). Dies bedeutet, daß man wahlfrei auf den verschiedenen Abstraktionsebenen arbeiten kann, die durch die Ebenen der Aufbaustruktur gebildet werden. Es kann beliebig zwischen 'Bottom-Up' und 'Top-Down'-Strategie gewechselt werden. Neu erzeugte Planungsobjekte sind immer Teilobjekt des im Focus befindlichen Objektes. Auch sie können später beliebig in der Aufbaustruktur ein-, um- und ausgebaut werden.

[5] Die Begriffe Focus, (Top-) Komplex, Simplex sind im Glossar im Anhang A erklärt.

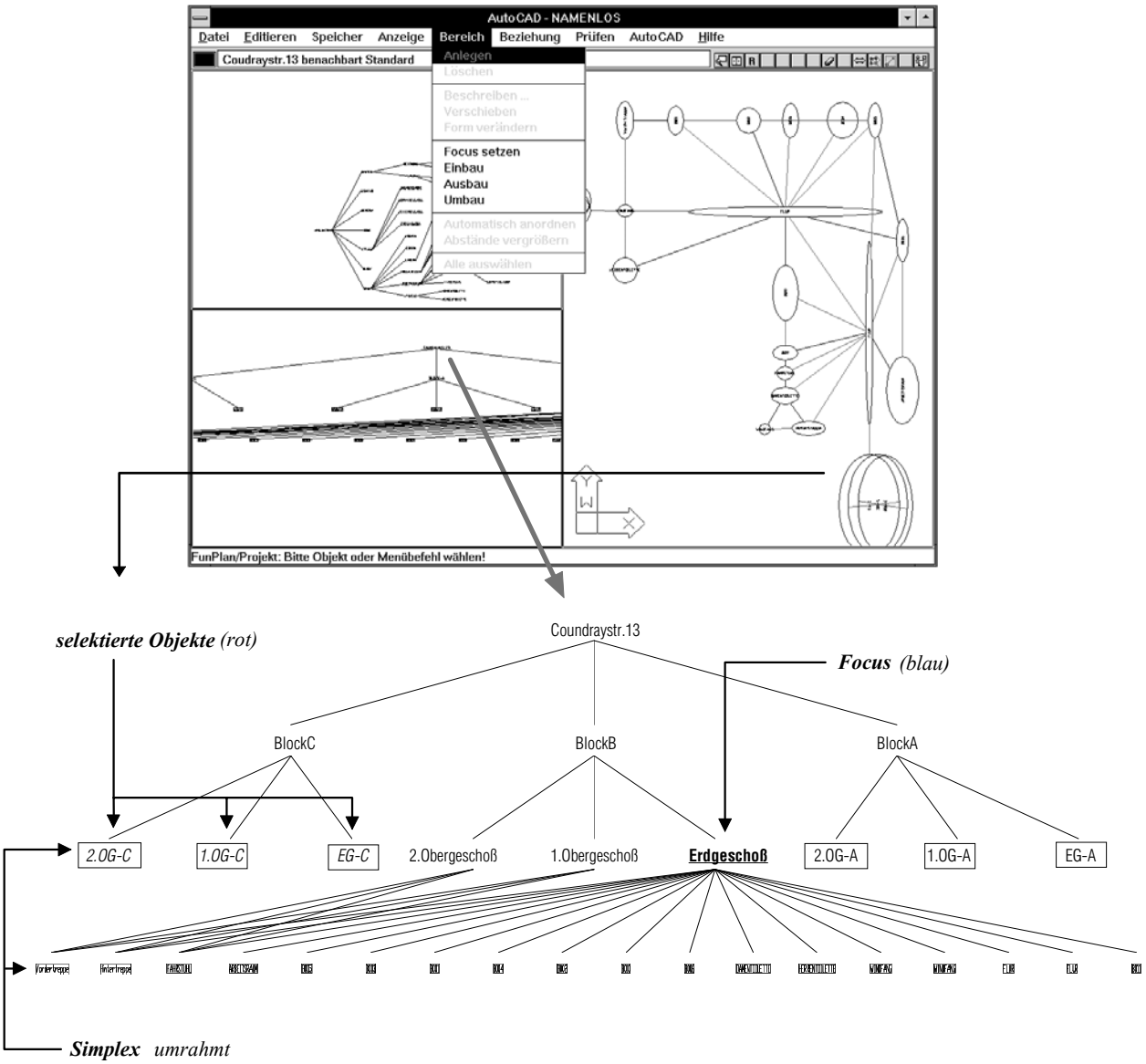
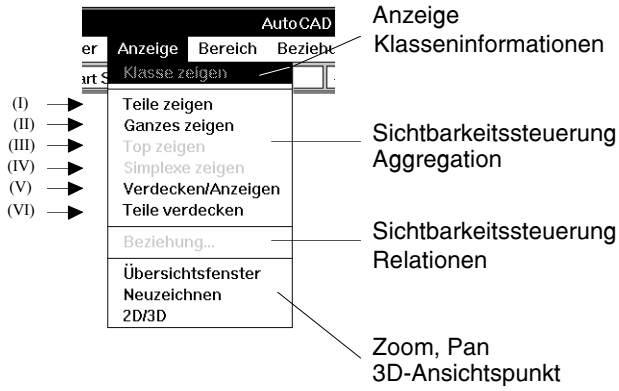


Abb. 7-18 Aufbaustruktur und Focussteuerung

Wenn CAD-Modelle über eine echte, durch den Nutzer zugreifbare Aufbaustruktur verfügen, tritt bei der grafischen Repräsentation das Problem der '*visuellen Inkonsistenz*' auf. So können Komplexobjekte durch ein geschlossenes Instanzobjekt repräsentiert werden, jedoch dual auch durch die Menge der Teilobjekte, die den Komplex bilden. Würde gleichzeitig das Ganze als auch die Menge der Teile grafisch repräsentiert, sähe man das Objekt gewissermaßen 'doppelt', d.h. die Ansicht wäre *visuell inkonsistent*. Da das Problem rekursiv für alle Teile auftritt, die selbst Komplex sind, käme keine geordnete grafische Darstellung zustande. Für FUNPLAN wurde ein Satz sichtbarkeitssteuernder Funktionen erarbeitet. Auch hier war Grundsatz, dem Nutzer größtmögliche Freiheit beim Modellieren zu gestatten. So kann die Anzeige für Planungsobjekte jeder Aggregationsstufe an- oder ausgeschaltet werden, auch wenn die Gesamtanzeige dadurch visuell inkonsistent wird. Einige Befehle führen im Bedarfsfall jedoch die Anzeige in einen konsistenten Grundzustand zurück (Abb. 7-19 a-g).

Anzeigemenü



Simplexe zeigen (IV)

Alle Bereiche, die gegenwärtig Simplexe sind, werden angezeigt – alle anderen Objekte werden unsichtbar. Dieser Befehl stellt immer den Zustand der visuellen Konsistenz her.

Ganzes zeigen (II)

Für den selektierten Bereich wird das Objekt angezeigt, in das es als Teil eingebaut ist. Für das nun angezeigte Objekt werden alle direkten und indirekten Teile ausgeblendet.

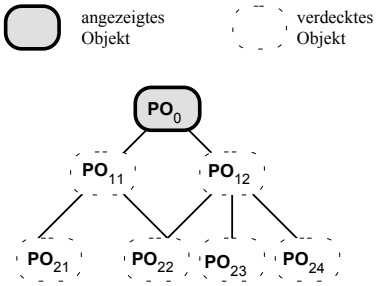
Teile zeigen (I)

Alle Teile eines selektierten Objektes werden angezeigt, das Objekt selbst wird verdeckt.

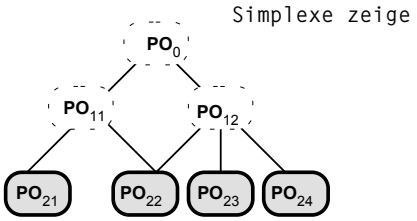
Verdecken/Anzeigen (V)

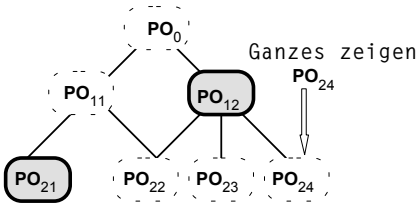
Ändert die Sichtbarkeit eines Objektes (Anzeige im Funktionsplan). Diese Funktion orientiert sich nicht an der visuellen Konsistenz.

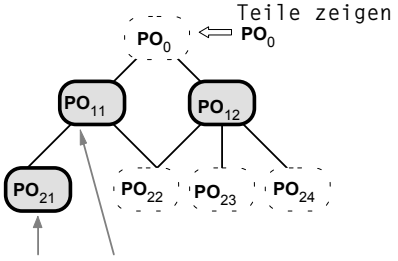
Beispiel eines Aggregationsgraphen



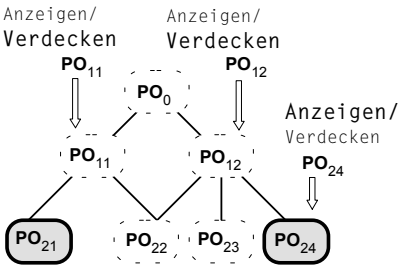
visuell konsistenter Ausgangszustand





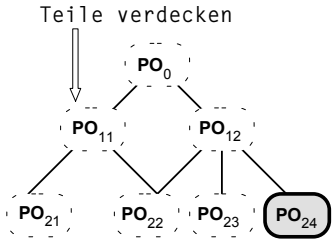


(Teil und Ganzes sichtbar visuell inkonsistent)



(visuell konsistent)

Teile verdecken **(VI)**
Dieser Befehl macht alle Teilbereiche des gewählten Komplexobjekts unsichtbar.



Top zeigen **(III)**
Durch diesen Befehl wird der zuerst angelegte Funktionsbereich angezeigt - alle anderen Bereiche (Objekte) werden unsichtbar. Dieser Befehl stellt immer den Zustand der visuellen Konsistenz her.

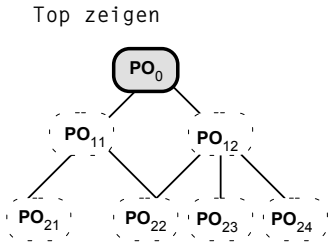


Abb. 7-19 a-g Veränderung der Sichtbarkeit von Entwurfsobjekten durch Steuerung im Aggregationsgraphen

Häufig sind einfache kaufmännische Auswertungen über Funktionspläne gewünscht, die auf der Auswertung des Aggregationsgraphen beruhen. Beispiel hierfür sind Flächenbilanzen (Summation über alle Simplexe) oder Preisabschätzungen durch Summation von Produkten aus 'Fläche' und 'Preis_pro_Flächeneinheit' aller Simplexe.

- CALCULATOR

Für ein selektiertes Objekt können Summen bzw. Produktsummen frei wählbarer numerischer Merkmale über Aggregate berechnet werden.

Summe S_I für eine Planungsobjekt I über Merkmal m :

$$S_I = \sum_{k=1}^{n} c_k \text{ mit}$$

$c_k = m_k$ wenn k Simplexe
$c_k = S_k$ wenn k Komplexe

Produktsumme P_I für ein Planungsobjekt I über Merkmal m_1 und m_2 :

$$P_I = \sum_{k=1}^{n} c_k \text{ mit}$$

$c_k = m_{k,1} \cdot m_{k,2}$ wenn k Simplexe
$c_k = P_k$ wenn k Komplexe

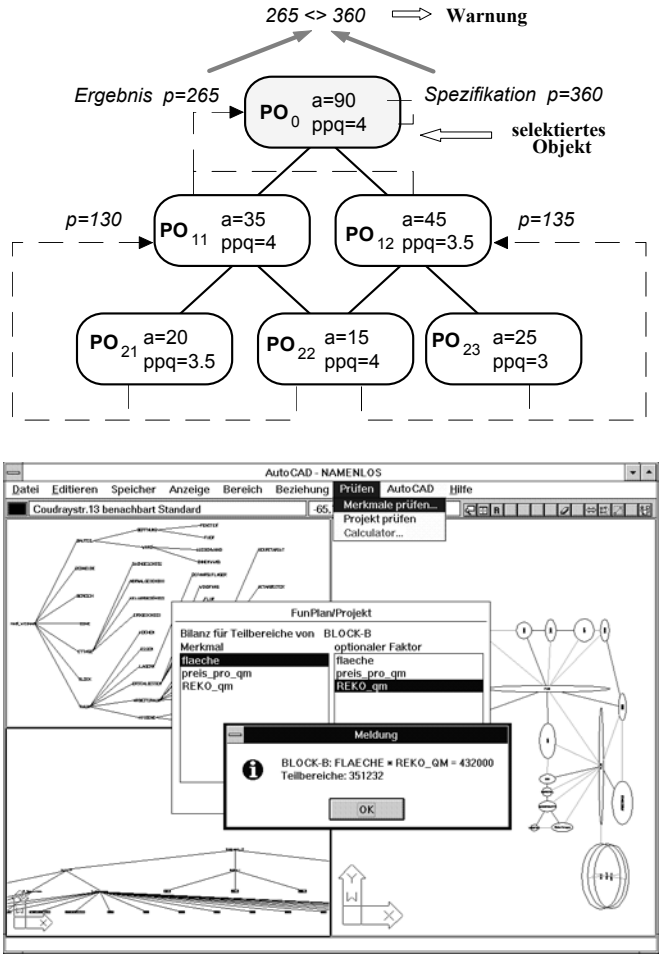


Abb. 7-20 Kummulative Berechnungen in Aufbaustrukturen

Auch hier tritt wieder die Dualität von Unschärfe und Constraints zu Tage. Wird beispielsweise in einem Objekt eine (unscharfe) Fläche spezifiziert, muß die Summation aller Teilflächen diesem Wert (etwa) entsprechen. Insbesondere im Top-Down-Entwurf kann dies jedoch nicht gesichert werden, da ein Komplexobjekt ja erst nach einer ganzen Sequenz von Entwurfsaktionen vollständig spezifiziert ist. Da diese Sequenz in FUNPLAN beliebig unterbrochen werden kann, differieren zwangsläufig Spezifikation und Summation zeitweise. Das System quittiert dies mit einer Warnung (Abb. 7-20).

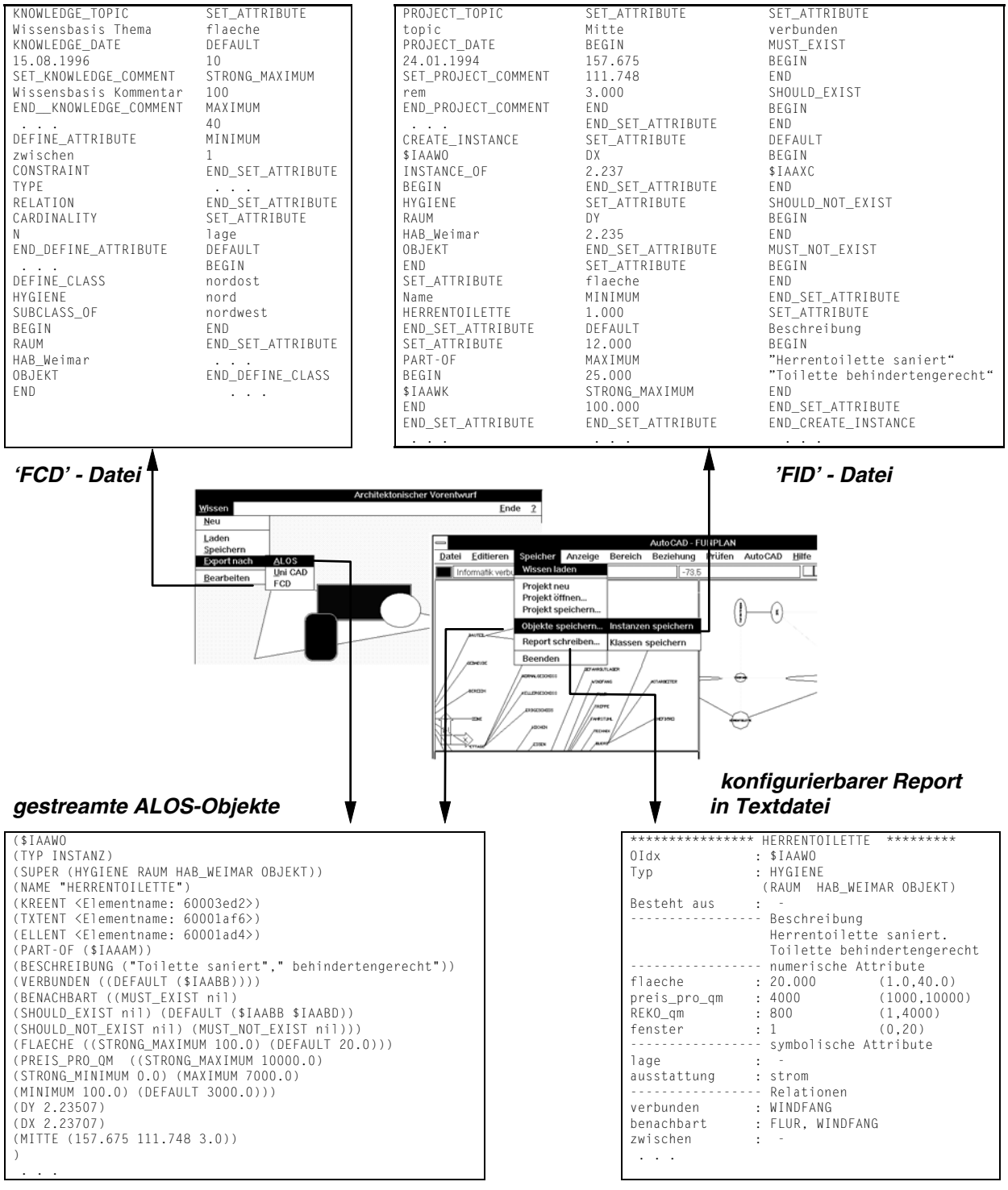


Abb. 7-21 Möglichkeiten zum Datenaustausch in FUNPLAN

- DATENAUSTAUSCH/ DATENSICHERUNG

In FUNPLAN bestehen zum Export der Pläne verschiedene Möglichkeiten. Durch die Implementation unter AutoCAD steht dessen volle Funktionalität zum Zeichnungsaustausch und zum grafischen Editieren der Funktionspläne zur Verfügung. AutoCAD-Funktionalität wird auch genutzt, um die Pläne zu drucken bzw. zu plotten. Bei der Benutzung der Zeichnungsschnittstelle geht jedoch jegliche Modellsemantik verloren. Eine weitere Möglichkeit besteht in der Nutzung der Funktionalität der objektorientierten AutoLisp-Erweiterung ALOS, die die Implementierungsgrundlage von FUNPLAN ist (siehe Kap.8, bzw. Anhang E). Alle ALOS-Objekte (Implementationsobjekte) können gestreamt[6] werden und stehen so anderen ALOS-Applikationen zur Verfügung. Dies betrifft auch die Klassenobjekte. Ein Datenaustausch über die DDE-Schnittstelle unter Windows ist ebenfalls realisierbar. Die Nutzung der FUNPLAN spezifischen Fileformat *FCD*[7] und *FID*[8] ist allerdings der beste Weg. Diese rein ASCII-basierte Schnittstelle orientiert sich strukturell an STEP [STEP92]. Sie ist objektorientiert strukturiert, unterstützt darüber hinaus sowohl den Transfer von Klassenobjekten ('FCD'-Format), das Facettenkonzept, als auch Erbschaftspfade[9] . Die Schnittstelle ermöglicht somit einen möglichst umfassenden Transfer von Modellinformationen.

In der unterstützten Planungsphase haben auch textbasierte Modellrepräsentationen ihren Platz. Hierzu wurde ein einfache Reportfunktion implementiert. Für ein Objekt und alle seine Teilobjekte können steuerbar alle Attribute und Relationen in eine ASCII-Datei ausgegeben werden. Diese Funktion bzw. die erzeugten Dateien können als Grundlage eines Raumprogramms gemäß DIN 276/277 dienen.

[6] überführen der ALOS-Objekte in ihre 'printed form', die sequentiell in eine ASCII-Datei geschrieben werden
[7] 'file of classdefinition' – Wissensbasis, siehe Anhang C
[8] 'file of instance definition' – Projektdaten, siehe Angang C
[9] gespeichert wird nicht nur die Klasse/Superklasse eines Objekts, sondern der Pfad im hierarchischen Klassengraphen bis zum standardisierten MetaObjekt 'OBJEKT'

7.3 INFPLAN

Um in PREPLAN Beschreibungselemente verfügbar zu machen, die nicht der gewählten Modellformalisierung genügen, können diese entweder in das CAD-Modell integriert werden (enge Kopplung) oder in einem eigenständigen Modell implementiert werden. Die Verknüpfung mit dem CAD-Modell erfolgt in diesem Fall durch eine ausgezeichnete Relation '*description_for*' nach dem Konzept der losen Kopplung.
Die enge Kopplung hat den Vorteil der nahtlosen Integration informaler Beschreibungen, etwa durch das Konzept der Attributtypklassen, die um Attribute vom Typ '*Dokument*' und dessen Subtypen zu erweitern wären sowie der Relationentypklassen für die benötigten Linktypen. Der Nachteil ist, daß die gesamte Funktionalität zum Retrieval und Authoring von informalen Beschreibungen dann Teil der CAD-Applikation wäre.

Für das angestrebte, in PREPLAN universell nutzbare Tool, wurde deshalb wiederum die lose Kopplung über die spezielle Relation '*description_for*' gewählt (Abb. 7-22).

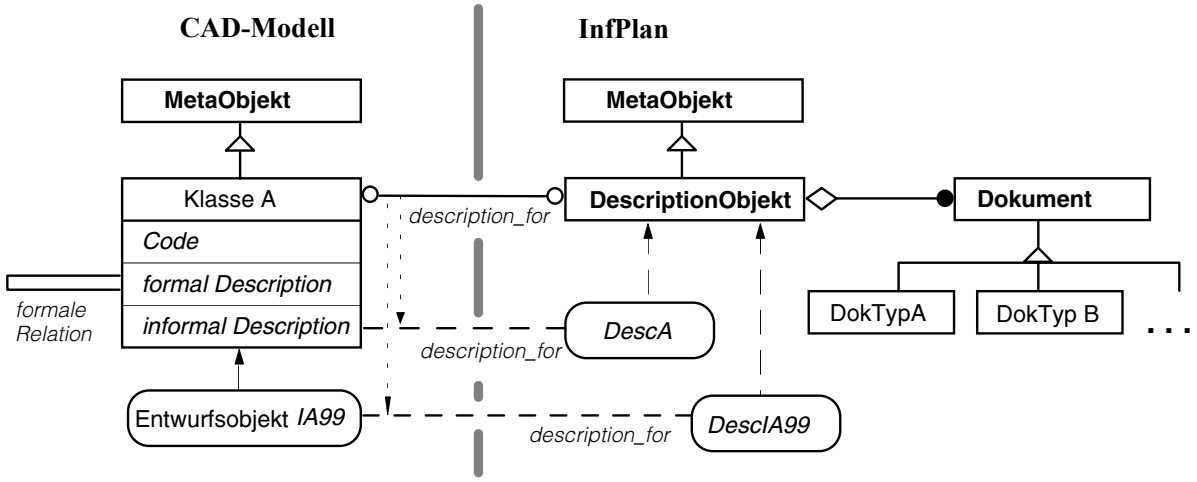


Abb. 7-22 Kopplung formale Planungsobjekte - informale Beschreibungen

Im Prototyp INFPLAN wurde mit folgender Funktionalität experimentiert:

- Das Retrieval nichtformalisierter Informationen aus dem CAD-Kontext heraus.
- Das Erzeugen und Editieren solcher Informationen.
- Das Strukturieren derartiger Informationssammlungen gemäß des formalen CAD-Modells.

Zur Realisierung dieser Funktionalität wurden exemplarische Lösungen mittels verschiedener Software entwickelt.

- *Variante 1* – DOS-Tool LINKWAY

LINKWAY bietet seitenorientierte Hypertexte. Mittels HotLinks wird ein Netz solcher Seiteninformationen strukturiert. Jedes CAD-Tool verwaltet je Objekt (Klassen und Instanzen) eine Einsprungseite, formalisierte CAD-Relationen werden nicht automatisch in LINKWAY übernommen. LINKWAY bietet Authoring - Funktionalität, so daß Hypertexte zur Laufzeit editiertbar sind. Die Konsistenz von CAD-Modell und Hypertext ist durch den Nutzer zu sichern.

- *Variante 2* – Windows-Help

Die Windows Hilfe bietet seitenorientierte Hypertexte, die um OLE[10] -Objekte erweitert werden können, wodurch dann beliebig-mediale Informationen verwendbar sind. Das Netz wird wiederum durch HotLinks strukturiert. Wie in Variante 1 verwaltet jedes CAD-Tool je Objekt eine Einsprungseite, formalisierte CAD-Relationen sind nicht durch automatisierte Übernahme aus dem CAD-Modell verfügbar. 'Windows Help' bietet zur Laufzeit keine Authoring-funktionalität. Alle Seiten werden mit einem RTF[11] - und bei Bedarf OLE-fähigen Editor erstellt und müssen anschließend mit einem 'Help'-Compiler in das '*.HLP'-Format übersetzt werden. Diese Technologie eignet sich gut zur Entwicklung von Informationssammlungen, die längere Zeit stabil bleiben und durch Nutzer nicht editierbar sind (statisches Klassenwissen). Auf Projektniveau kann sie nicht verwandt werden.

- *Variante 3* - TOOLBOOK

TOOLBOOK gestattet die Entwicklung objektorientierter Hypermediasysteme. Das System integriert vielfältige Medien- bzw. Dokumententypen über die OLE-Technologie unter MS Windows. Es verfügt über eine Scriptsprache zur Programmierung von Anwendungen sowie die unter Windows standardisierten Mittel zur Prozeßkommunikation, wie DDE ([TOO95]).
Im verfügbaren Prototyp stellt INFPLAN[12] eine OLE-Containerverwaltung in TOOLBOOK dar (Abb. 7-23). Ein Container ('*DescriptionObjekt*') enthält alle Dokumente, die mit einem CAD-Objekt assoziiert sind. Als einzige Relationen (und damit als Links) werden die im OO-Metamodell fixierten Relationen 'super'/ 'subtype', 'instance' und 'part_of'/'has_part' direkt nach INFPLAN exportiert, alle anderen (formalisierten) CAD-Relationen werden bei jedem Retrievalprozeß neu analysiert und zum Aufbau der entsprechenden Links nach TOOLBOOK exportiert. INFPLAN arbeitet 'stand-alone' nur über diese paradigmatischen Relationen. Durch die OLE-Technologie stellen die Serverapplikationen die Funktionalität zum Authoring der Dokumente zur Verfügung. Sie sind gegenwärtig nicht als Einzeldokumente verfügbar, können aber über die Windows-Zwischenablage ausgetauscht werden.

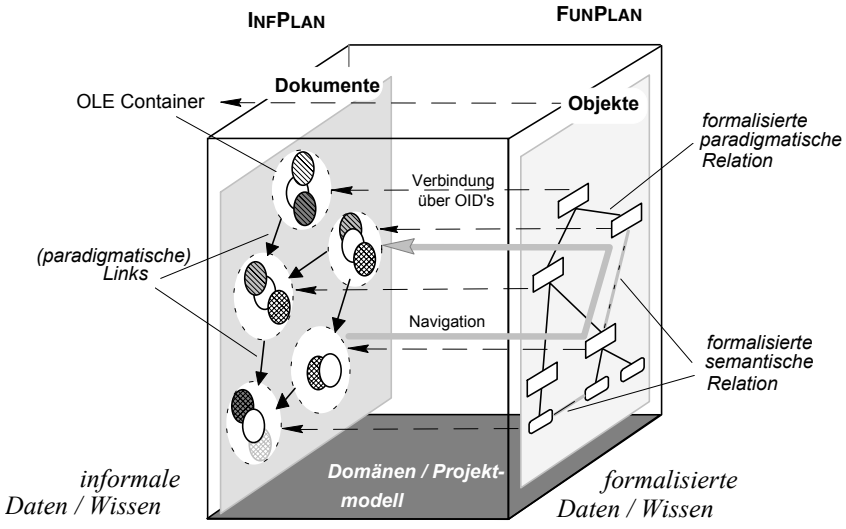


Abb. 7-23 Kooplung INFPLAN - FUNPLAN

[10] 'Object Linking and Embeding', eine Windows-Technologie zur Dokumentenintegration
[11] Rich Text Format - ein neutrales Dokumentenformat
[12] Der Prototyp wird ausführlich beschrieben in [SKE95]

Zur technischen Kopplung der Module wurde der DDE-Mechanismus verwendet, über ihn wird die Funktionalität von INFPLAN den anderen Tools zur Verfügung gestellt. Die implementierte Funktionalität ist somit auch bei einer 'stand alone' Verwendung von INFPLAN verfügbar. Auf eine Verwendung von INFPLAN aus dem Projektteil heraus wurde auf Grund des hohen Ressourcenbedarfs verzichtet. Die im Modul 'FUNPLAN-Wissen' verwendeten Datenstrukturen und Steuermechanismen sind aber grundsätzlich auch unter AutoCAD verfügbar, eine Kopplung nach demselben Prinzip also leicht realisierbar.
In INFPLAN sind einerseits Funktionen als Interface für rufende Applikationen definiert, andererseits müssen in der rufenden Applikation bestimmte Funktionen realisiert werden, um INFPLAN den Zugriff auf das formale CAD-Modell zu ermöglichen. Diese Art der Kopplung wurde mit dem KappaPC basierten Modul FUNPLAN-Wissen getestet. Folgende Gruppen von Funktionen können durch Applikationen genutzt werden:

- *Funktionen, die den Dialog an INFPLAN übergeben*
  `IPShow(obj, relations)` - Das 'DescriptionObjekt' für '`obj`' wird angezeigt, kann aber nicht editiert werden. Die Liste '`relations`' überträgt die als Links verfügbaren formalen Relationen
  `IPWork(obj, relations)` - Wie `IPShow`, aber das 'DescriptionObjekt' für '`obj`' (ein Dokumente) ist über Serverapplikationen editierbar

- *Funktionen ohne Dialog in INFPLAN (Systemsteuerung)*
  `IPStart()` / `IPEnd()` - Starten INFPLAN, Initialisieren Verbindung, bzw. Beenden INFPLAN
  `IPLoad(filename)` / `IPSave(filename)` - Laden/Sichern einer Dokumentensammlung
  `IPNew(rootname)` / `IPDelete()` - Erzeugen einer neuen, leeren Dokumentensammlung mit '`rootname`', löschen einer Dokumentensammlung

- *Funktionen, die ohne Dialog in INFPLAN zur Übernahme von Änderungen des CAD-Modells (Modellveränderungen) dienen*
  `IPCreateClass(newobj, parent)` / `IPCreateInstance(newobj, parent)` - Erzeugen eines neuen, leeren 'DescriptionObjekts' (OLE-Container)
  `IPRenameObject(oldname, newname)` / `IPDeleteObject(obj)` - Löschen / Umbenennen eines 'DescriptionObjekts'

Folgende Funktionen müssen durch die rufende Applikation bereitgestellt werden:

- *Funktionen, die INFPLAN die Analyse der Struktur des CAD-Modells erlauben*
  `GetParent(obj)` - liefert die Klasse bzw. die Superklasse von '`obj`'
  `GetSub(obj)` - liefert die Subklassen bzw. Instanzen von '`obj`'
  `GetRel(obj)` - liefert die Liste der formalen Relationen für '`obj`'
  `GetRelObjects(obj, slot, facet)` - liefert die Liste der Objekte, die über eine formale Relation '`slot`' für '`obj`' assoziiert sind.

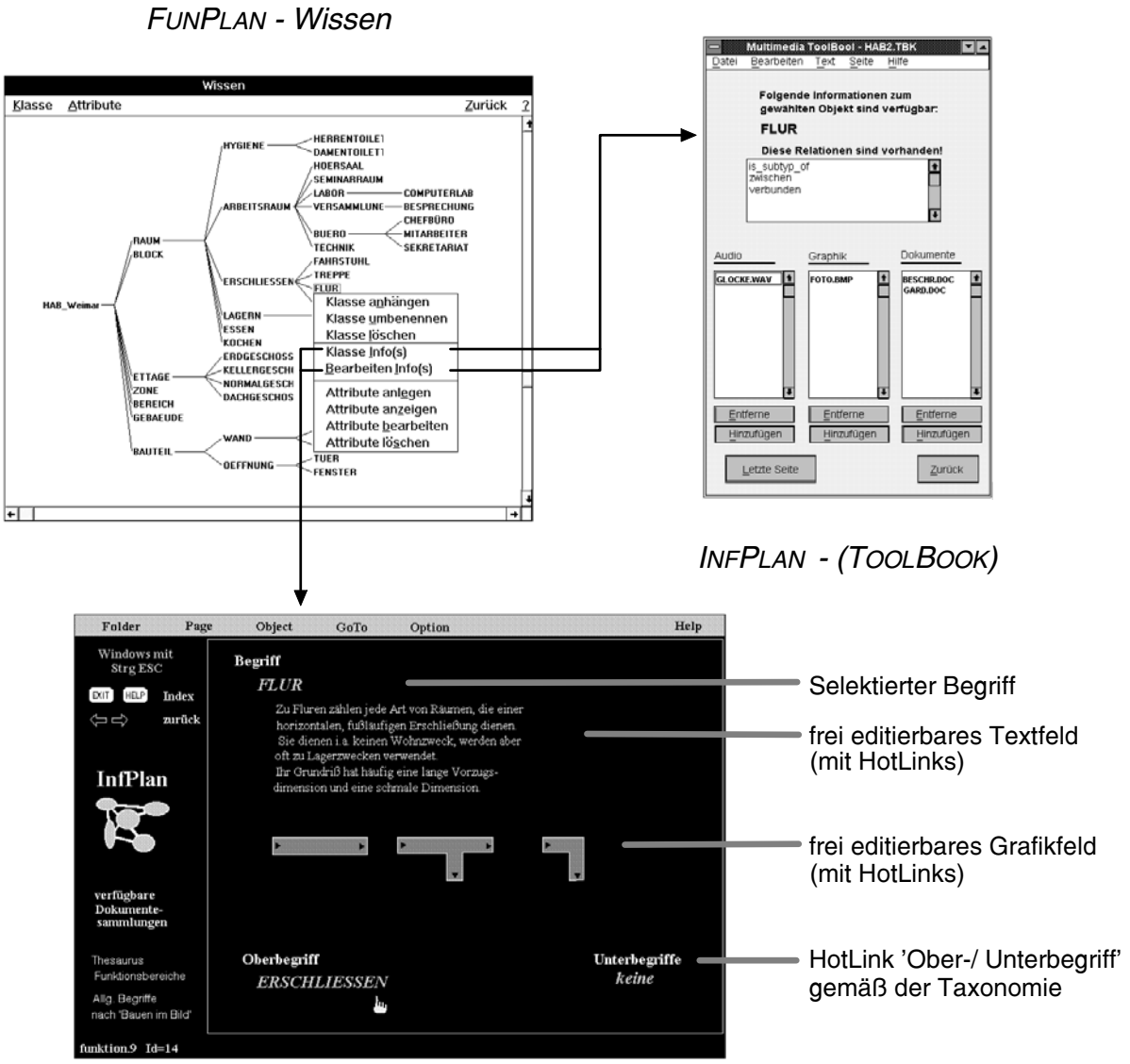


Abb. 7-24 INFPLAN als TOOLBOOK bzw. LINKWAY-Implementierung

In der gegenwärtigen Variante sind die Dokumentensammlung selbst nur für die paradigmatischen Relationen (Links) strukturiert. Jede weitere Strukturierung liegt formal nur im Projektmodell vor. Für eine Weiterentwicklung erscheinen folgende Erweiterungen dieses Konzeptes sinnvoll:

- Dokumentensammlung vollständig strukturiert, formale Relationen auch informal als HyperLinks verfügbar, Dokumente sind terminale Knoten (nur Links auf OLE-Container, Abb. 7-25a)

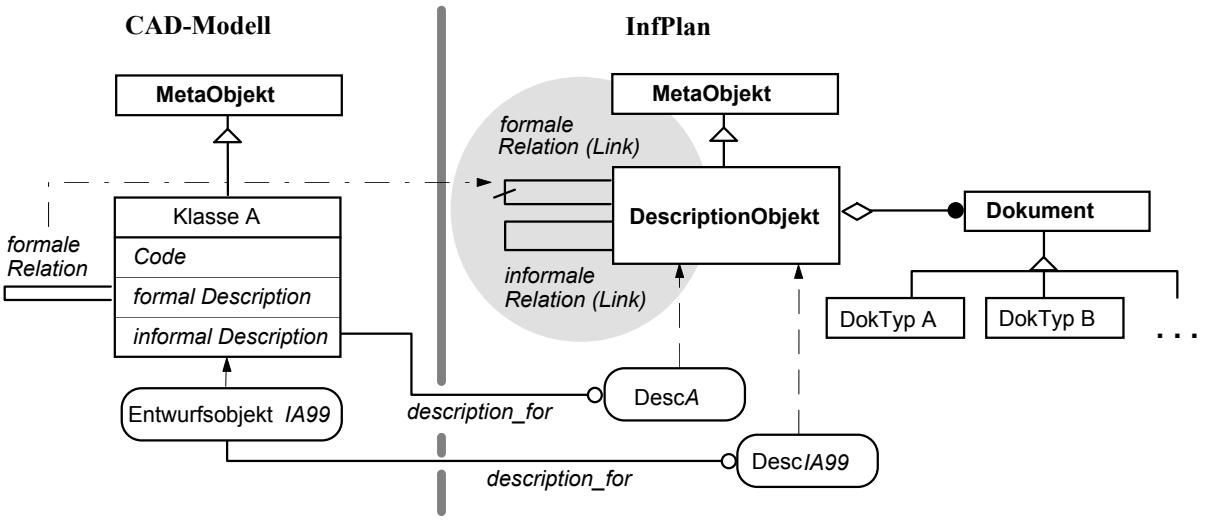


Abb. 7-25 a Implementationsvariante INFPLAN

- Dokumentensammlung vollständig strukturiert, Strukturierung durch formale Relationen auch informal als HyperLinks,
  Dokumente sind nichtterminale Knoten, d.h. rekursiv aufgebaute Dokumente (Links auch auf Einzeldokumnte in OLE-Containern, Abb. 7-25b).

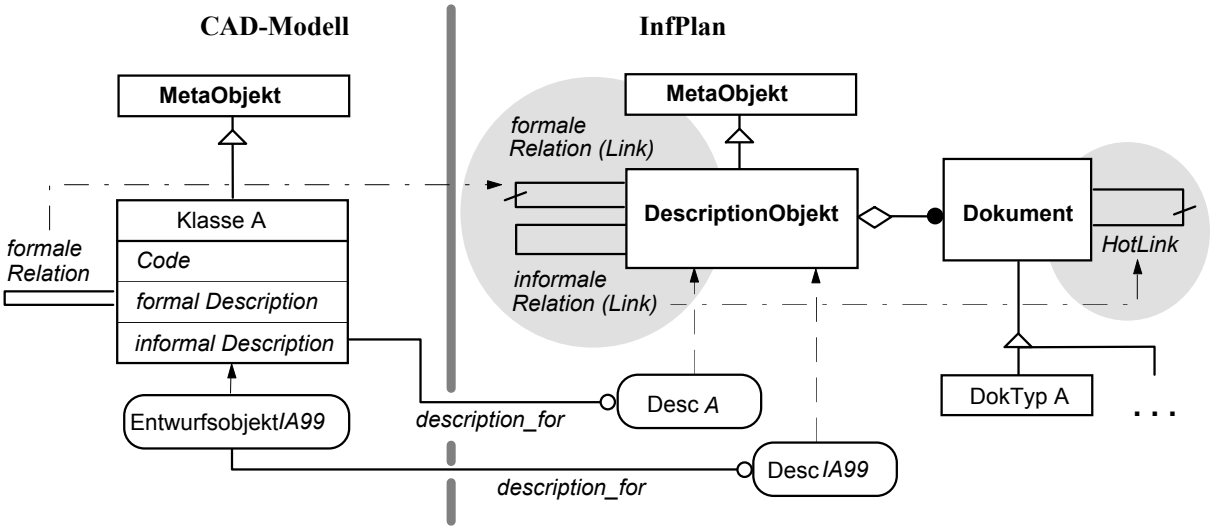


Abb. 7-25 b Implementationsvariante INFPLAN

Alle angeführten Varianten gehen von einem ‘starken’ CAD-Modell aus, das letztlich die Struktur und damit das Retrievalverhalten einer solchen informalen Erweiterung eines CAD-Produkt- bzw. Domänemodells bestimmt. Für langlebige und umfassende Modelle, die beispielsweise eine weitgehende Büroautomation einschließen, läßt sich allerdings nur schwer die Konsistenz von CAD-Modellen und informalen Informationssammlungen sichern. Zukünftige Lösungen sollten diesen Fakt *anerkennen*! Dies kann geschehen, indem für Retrievalprozesse sowohl im formalen doch vor allem im informalen Modell Anfragemodelle aufgebaut werden, wie sie von Informationssystemen bekannt sind. Jede Anfrage wird dort als zweckabhängiger Interpretationsprozeß verstanden, die sie ja tatsächlich ist. Diese

Technologie geht von einem vollständig dynamischen Weltmodell aus, dessen Strukturierung und Analyse auf der Basis erkenntnistheoretischer Betrachtungen geleistet wird.

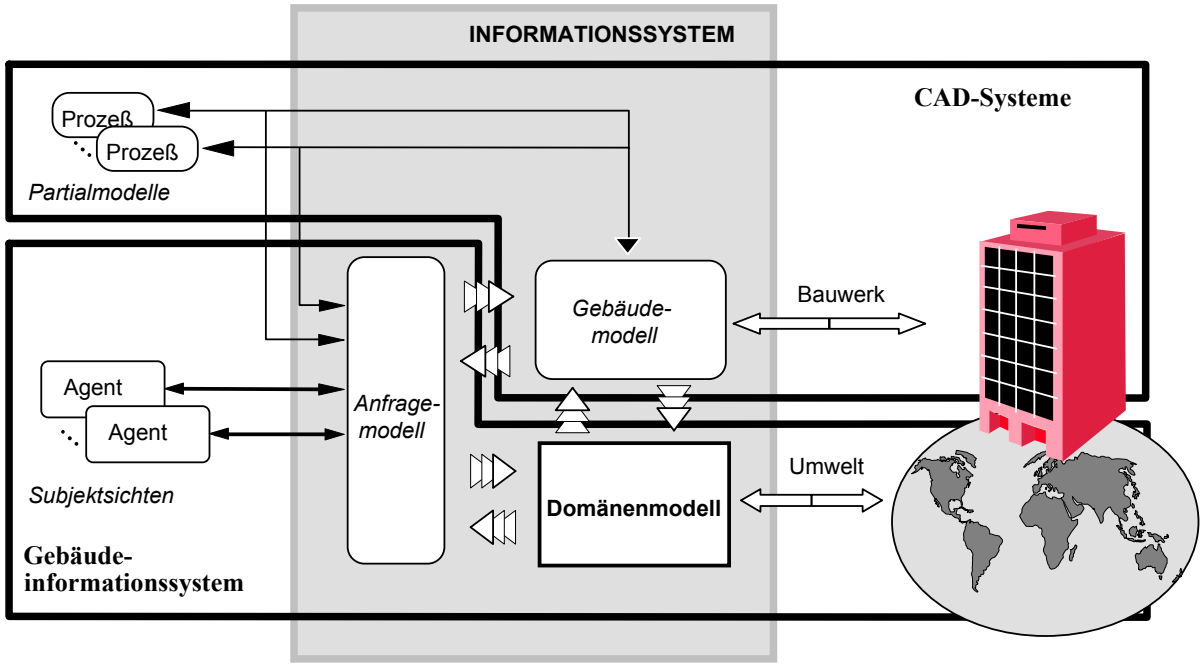


Abb. 7-26 Anfragemodelle als Basis umfassender Gebäudeinformationssysteme

7.4 NETGEN – Variante eines Entwurfsgenerators

Der mittels FUNPLAN erstellte Graph aus Planungsobjekten (Knoten) und deren Beziehungen (Kanten) spezifiziert eine Topologie, die für die Grundrißgeometrie gültig sein soll. Selbst bei der direkten Arbeit auf der grafischen Repräsentation der Blasendiagramme, wie sie in FUNPLAN realisiert wurde, ist das Mappen der Topologie auf eine (Graphen-) Geometrie ein nicht-eindeutiger abduktiver Prozeß (Abb. 7-27). Auch wenn grafisch spezifiziert wird, sind die erzeugten geometrischen Informationen nur eine vorläufige Variante. Dies ist ein weiterer Grund, warum Funktionsplanskizzen gegenwärtig nicht archiviert werden – ihre grafischen Informationen stimmen häufig nicht mehr mit den Plänen überein, die das Ergebnis der Formfindung und Konstruktion sind.

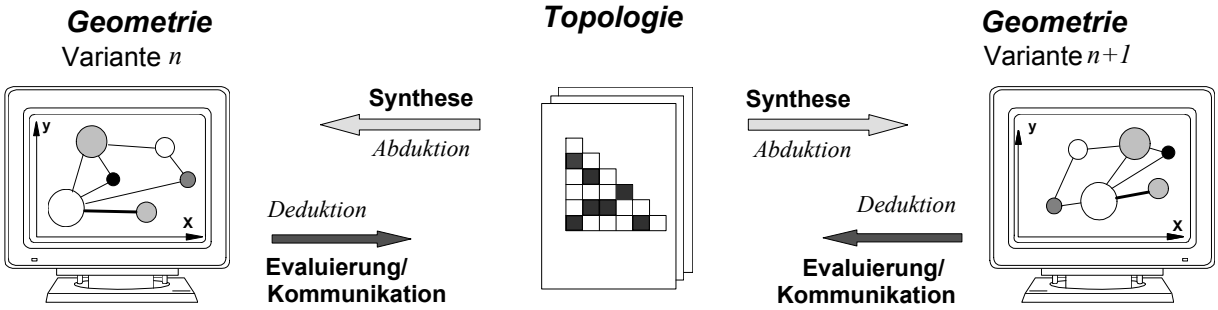


Abb. 7-27 Nichteindeutige Transformation Topologie ⇔ Geometrie

Durch die grafische Arbeitsweise verfügen in FUNPLAN Funktionsobjekte bereits über eine Position auf der Zeichenfläche. Diese Positionen sind durch NETGEN unter Maßgabe von Zielfunktionen zu variieren. NETGEN gehört damit in die Klasse der modellvariierenden Entwurfsautomaten (siehe Kapitel 2). Auch wenn hier in einfacher Weise Aspekte der Formfindung berührt werden, erfolgt bestenfalls eine Aufarbeitung des Graphen in einer Art, die den Formfindungsprozeß unterstützt. Es wird einerseits eine plausible Anfangslösung für den abduktiven Suchprozeß gebildet wird. Andererseits wird die Evaluierung einer Kubatur durch die übersichtliche Darstellung des Funktionsgraphen erleichtert.

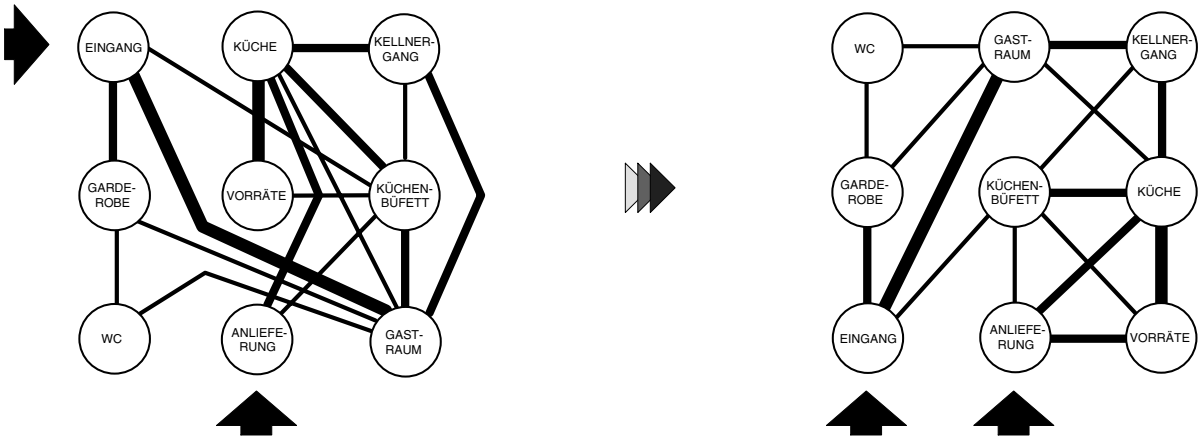


Abb. 7-28 Varianten desselben Funktionsgraphen nach HEUSER (aus [HEU89])

Das Problem der Erzeugung einer Graphengeometrie aus einem gegebenen Relationennetzwerk kann wie folgt beschrieben werden:

Gegeben ist der Graph G des Relationennetzwerkes. Die Objekte o_i stellen die Elemente der Knotenmenge $O = \{o\}$ dar, die Relationen k_k werden durch die Kantenmenge $K = \{k(o_i, o_j)\}$ mit $i \neq j$ gebildet. Gesucht werden die Positionen $\overline{p}_i^T$ der Knotenobjekte o_i in einem gegebenen begrenzten Gebiet $\Re$, mit folgenden Zielfunktionen

(1) Das Gebiet soll möglichst gleichmäßig mit Knoten gefüllt sein.

(2) Es soll eine möglichst geringe Anzahl sich kreuzender Kanten entstehen.

Prinzipiell handelt es sich hier um eine Optimierungsaufgabe. Deren Lösung bereitet mit Standardmethoden allerdings Schwierigkeiten, weil

- es sich hier um eine Polyoptimierung handelt, deren Zielfunktionen (1) und (2) partiell konkurrierend sind (wie sich am Beispiel zeigen läßt)
- das Problem in die Klasse der gemischt ganzzahligen Optimierungen gehört
- eine mathematische Formulierung des Anspruchs (1) schwer fällt.

Auch für größere Probleme (Objektzahl $n>50$, Relationenzahl $m \sim 2*n$) soll auf PC's in angemessener Zeit ($t<60sec$) eine gute Näherungslösung gefunden werden.

Das beschriebene Problem tritt in ähnlicher Form bei der Organisation des Nervensystems höherer Lebewesen auf. Die Zuordnung von Rezeptorzellen zu Zellarealen in der Großhirnrinde, in denen die Verarbeitung rezipierter Reize erfolgt, ist nicht genetisch fixiert ([RIT93]). Diese Zuordnung muß in einem Selbstorganisationsprozeß erlernt werden. Der Abbildungsprozeß heißt '*neuronale Projektion*' und weist die Eigenschaft auf, daß er 'nachbarschaftserhaltend' ist, d.h. daß benachbarte Elemente der rezipierenden Schicht auf benachbarte Elemente der abbildenden Schicht gemapt werden. Die Abbildung füllt weiterhin das Darstellungsgebiet ganz aus, d.h. es gibt keine Hirnareale, die funktionslos sind.

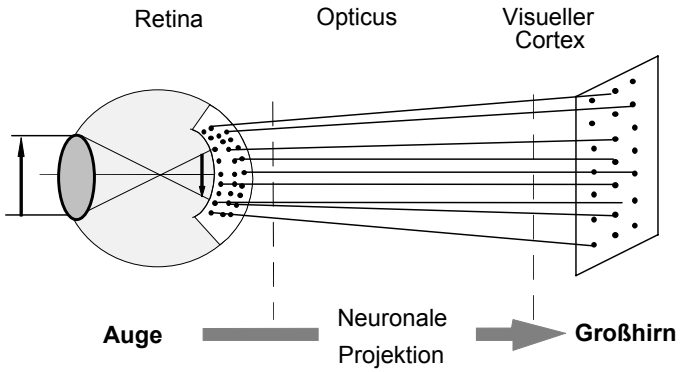


Abb. 7-29 nachbarschaftserhaltende Abbildung Retina - Hirnrindenfeld

Diese Fähigkeit zur Selbstorganisation wird mit Neuronalen Netzen simuliert, wie sie von KOHONEN (u.a. in [MAR90]) entwickelt wurden. Sie enthalten dort folgende Vereinfachungen:

- Das Hirnareal wird durch eine Matrix W repräsentiert, wobei jedes Matrixelement $w_{i,j}$ ein Neuron darstellt.

- Die Rezeptorfläche wird durch eine Region der Größe $0 \le x \le 1$ und $0 \le y \le 1$ dargestellt, Reize sind Vektoren in dieser Fläche $[x,y]^T_{Reiz}$.
- Jedes Neuron $w_{i,j}$ speichert einen 2D-Vektor $[x,y]^T_{i,j}$, der die Stelle der Rezeptorfläche markiert, für die dieses Neuron am 'zuständigsten' ist, d.h. für das $\min\left|[x,y]^T_{i,j},[x,y]^T_{Reiz}\right|$ gilt.

Gesucht ist eine Belegung der Neuronen $w_{i,j}$ mit Vektoren $[x,y]^T_{i,j}$, so daß benachbarte Neuronen der Matrix benachbarte Vektoren der Rezeptorfläche speichern (Zielfunktion 1) und die Rezeptorschicht gleichmäßig mit Vektoren überlagert ist (Zielfunktion 2).

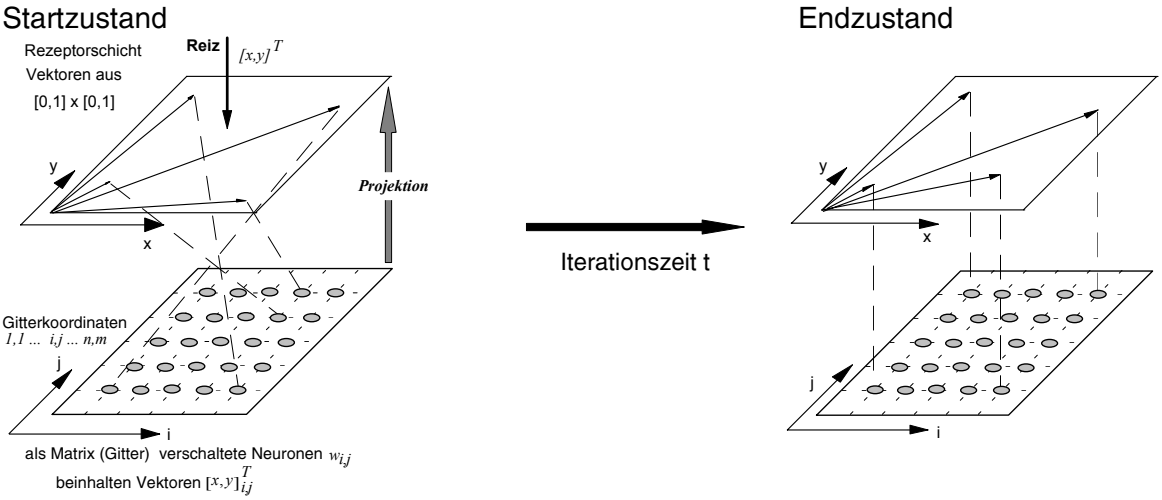
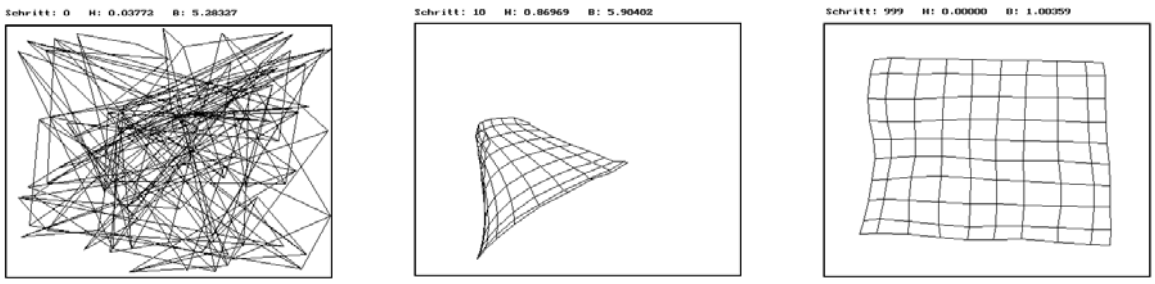


Abb. 7-30 a, b Selbstorganisation des Neuronennetzes

Dieser Netztyp lernt 'nonsupervised', d.h. ohne Eingabemuster. Da nur bestimmte zuständige Neuronen lernen dürfen, handelt es sich hier um 'competatives' Lernen (Kohonenregel). Der Algorithmus hierzu wurde bereits ausführlich in [STE94] vorgestellt.



ungeordnet partiell geordnet geordnet

Abb.7-31 a-c Entwicklungsphasen eines 10 x 10 Neuronengitters auf eine [0,1] x [0,1] Rezeptorschicht

Im Beispiel Abb. 7-31 wurde ein 2D-Gitter auf eine quadratische 2D-Fläche abgebildet, d.h. daß die abzubildende Graphstruktur ein Gitter war. Verallgemeinert man das Verfahren, so lassen sich beliebig dimensionale Topologien auf n-dimensionale Regionen abbilden[13] .

[13] eine ausführliche Darstellung des Programms erfolgt in Anhang D

Um dieses Verfahren jedoch auf den beschriebenen konkreten Fall anwenden zu können, mußten folgende Erweiterungen vorgenommen werden:

- Die in der Literatur beschriebenen Anwendungen des Verfahrens (etwa [RIT93], [KRU91], [DOR91]) wenden das Verfahren nur zum Maping regelmäßiger Strukturen wie Gitter, Ringe, Zylindergitter etc. an. Bei der vorliegenden Anwendung treten i. allg. keine regelmäßigen Topologien auf, sondern beliebig strukturierte Graphen. Die Matrixelemente $w_{i,j}$ werden hier durch die Knotenelemente o_i gebildet. Die implizite Nachbarschaftsrelation wurde durch explizite Kanten k_k ersetzt. Der Grundalgorithmus wurde dahingehend angepaßt, daß die Ermittlung der mitlernenden Neuronen durch Suche im Graphen erfolgt. Dadurch ist die Darstellung auch gerichteter Relationen kein Problem mehr (siehe Abb 7-32 a).
- An Stelle der [0,1] x [0,1] Reizflächen des Beispiels (Abb. 7-32 a und b) können auch beliebige Gebiete $\Re$ gewählt werden, so daß gilt $\overline{p}^T_{\mathrm{Reiz}} \in \Re$. So lassen sich beispielsweise die Knoten (Funktionseinheiten) in einem gegebenen ebenen Grundrißpolygon oder einem geraden Prisma mit dieser Grundfläche konzentrieren.
- Die Koordinaten des Gebietes $\Re$ müssen nicht dicht liegen, sondern können diskret verteilt sein. Grundvoraussetzung ist, daß sich ein skalarer Abstand für jedes Punktpaar des Gebietes bestimmen läßt. Ein Mappen auf 2½D Gebiete wird so möglich und zeigt gute Ergebnisse. So können z.B. die Funktionseinheiten in diskreten Höhensprüngen (Geschosse) über einem 2D-Schnittpolygon angeordnet werden (Abb. 7-32 c).
- Bei der Beschreibung von Funktionselementen in FUNPLAN ist es z.B. möglich, deren Lage im Grundriß (-polygon) festzulegen (z.B. Knoten : 'Terrasse', Lage : (Süd, Südwest)). Der Algorithmus sollte diese Lagespezifikation beachten. Entsprechend dem festgelegten Wertevorrat für das Attribut 'Lage' wurde dazu die Reizfläche in 16 Teilflächen geteilt (wie in Abb.7-32 b). Aus Flexibilitätsgründen werden die 9 Lagesymbole durch überlappende Teilflächen gebildet. Beim Maping wird dann eine Knotenverschiebung aus den zugewiesenen Teilflächen verhindert (siehe Abb. 7-32 d).
- Um bewertete Relationen abbilden zu können, wie sie in Funktionsplänen auftreten, wurden Verschiebungsfaktoren eingeführt, die die Verschiebungsradien im Lernprozeß der Kantenbewertungen anpassen. Das führt zu der Tendenz, daß für Knoten, die über 'starke' Kanten verbunden sind, eine Benachbarung bevorzugt wird. Gegenwärtig werden die Bewertungen '-', '+' unterstützt. Durch negative Kantenbewertungen können auch Abstoßungen realisiert werden (Abb 7-32 d).

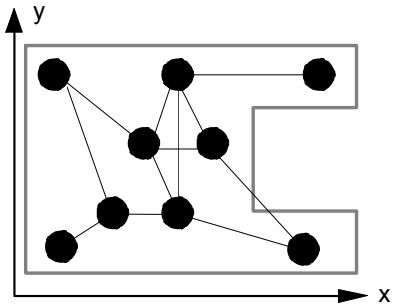


Abb. 7-32 a Maping eines Netzgraphen auf ein 2D-Polygon

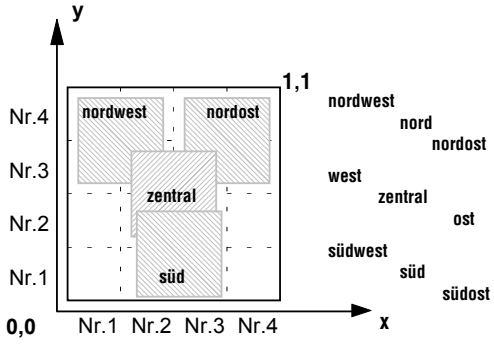


Abb. 7-32 b Orientierung im Grundriß durch symbolische Spezifikation

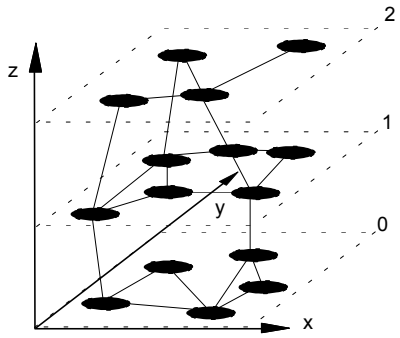


Abb. 7-32 c Maping eines Netzgraphen auf eine 2½D-Fläche

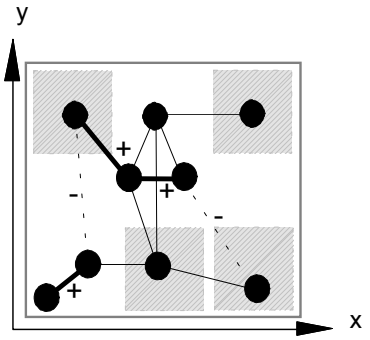


Abb. 7-32 d Maping eines Netzgraphen mit bewerteten Kanten auf ein 2D-Polygon, mit fixierten Knotenpositionen

Mit dem Verfahren gelingt die übersichtliche Visualisierung von Funktionsplänen. Selbst große Netze lassen sich mit akzeptablen Rechenzeiten visualisieren. Als Testbeispiel diente ein Relationennetzwerk mit 75 Knoten und 375 Kanten. Für die Ermittlung der Knotenkoordinaten benötigte ein PC (CPU 80486, 66 Mhz) 28 Sekunden. Der Algorithmus kann durch Vorgabe von Parametern in einer Konfigurationsdatei (siehe Anhang D), der Netzgröße und der erwünschten Abbildungsqualität flexibel angepaßt werden. Da die Abbildung Topologie auf Geometrie nicht eindeutig ist, wird eine zufällige Variante generiert. Durch Ändern der Zufallszahleninitialisierung in der Steuerdatei lassen sich beliebig oft neue Varianten generieren. NETGEN wird z.Z. in FUNPLAN unter Windows als Hintergrundprozeß eingebunden, die Kommunikation erfolgt im gegenwärtigen Prototyp über eine Dateischnittstelle (Abb.7-33).

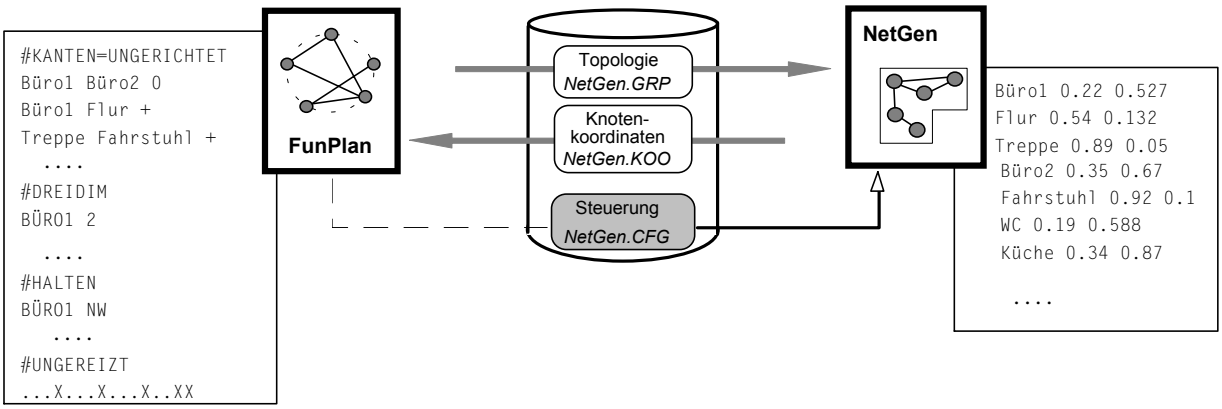


Abb. 7-33 Kopplung NETGEN - FUNPLAN

NETGEN zeigt, daß Generierungstools in CAD-Systemen Tools vom Typ ‘design assistant’ sinnvoll ergänzen können und mit einem ‘design repository’ gekoppelt werden können. Sie unterliegen allerdings den bekannten Einschränkungen der Softwareentwicklung. So ist NETGEN bedingt durch das spezifische Dateiformat nur mit FUNPLAN benutzbar und stellt Anforderungen zur Ausprägung der Domänen in der Form, daß

- Objekte über 2½D Lagekoordinaten verfügen müssen (Etagennummer ganzzahlig)
- ihre Orientierung im Grundriß symbolisch mit einer fixierten Menge von Symbolen spezifiziert werden muß (‘N’, ‘NO’, ‘O’, ‘SO’, ...)
- die Bewertung selektierter Relationen gegenwärtig nur symbolisch mit +, - erfolgen kann.

Dem Gedanken des '*design repository*' ist das nicht abträglich, da im Rahmen der Modellbewahrung und des Modellaustausches es nicht maßgeblich ist, mit welchem Tool die Modelle erzeugt wurden. Solange deren Interpretation nicht vom Vorhandensein des erzeugenden Tools abhängt oder aber die Interpretierbarkeit durch das Erweitern des Modells um toolinterne Daten eingeschränkt wird, ergänzen sie reine Modelleditoren wie FUNPLAN.
Durch die Möglichkeit, Grundrißpolygone und einfache prismatische Kubaturen über Grundrissen als Ausgabegebiet festzulegen, eignet sich NETGEN gut zur Variantengenerierung in der Redesignphase. Dieses Einsatzfeld scheint inhaltlich besonders attraktiv, da in diesem Fall ein oftmals sehr präzises Raumprogramm in einem gegebenen Grundriß unterzubringen ist. Die Lösung ist hier durch den Architekten in einem als wenig kreativ empfundenen Permutationsprozeß aus einer gegebenen Anordnung oder einer unzureichenden initialen Lösung zu erzeugen. In diesem Fall kommen die Stärken des Computers als 'number cruncher' zum Tragen, Ergebnisse werden durch den Nutzer akzeptiert.
Beobachtungen beim Einsatz des Systems in der Lehre und bei Messepräsentationen (CeBIT'95) zeigen, daß ein starkes Interesse gerade an diesem Tool besteht. Inwieweit dieses Interesse aber durch qualitativ bessere Entwürfe gerechtfertigt ist und nicht nur durch den optisch verblüffenden Effekt der Selbstorganisation bedingt ist, müssen weitere Untersuchungen zeigen.

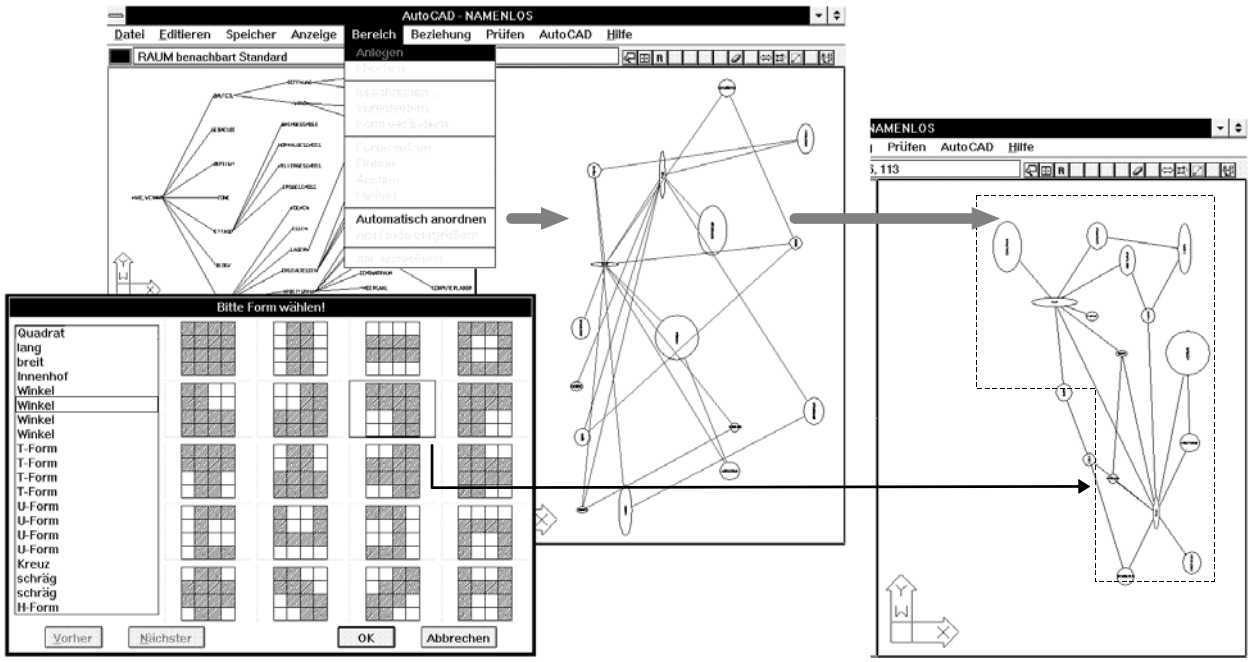


Abb. 7-34 Mit NETGEN optimierte Graphengeometrie

7.5 Systemarchitektur – Realisierung

Die bisher beschriebene Funktionalität zur Wissensbasierung erfordert für ihre Implementation ein System, das 'echte' Klassenobjekte kennt – nur dann sind Klassen auswertbar, veränderbar, erzeugbar sowie eine Analyse der Taxonomie möglich. Compilierende Standardprogrammiersprachen, wie C++, genügen diesen Anforderungen nicht, während viele objektorientierte KI-Shells über diese Merkmale verfügen. Die Implementation sollte auf PC-Basis erfolgen, da dies gegenwärtig die Standardausrüstung in Büros und auch in der

Ausbildung ist. Unter Windows 3.11 wurde die KI-Shell KappaPC ausgewählt, da sie die o.g. Anforderungen erfüllt. Sie verfügt allerdings nicht über ein Facettenkonzept, so daß dies über Extensionen der Attributnamen simuliert werden mußte (siehe Abb. 7-35).

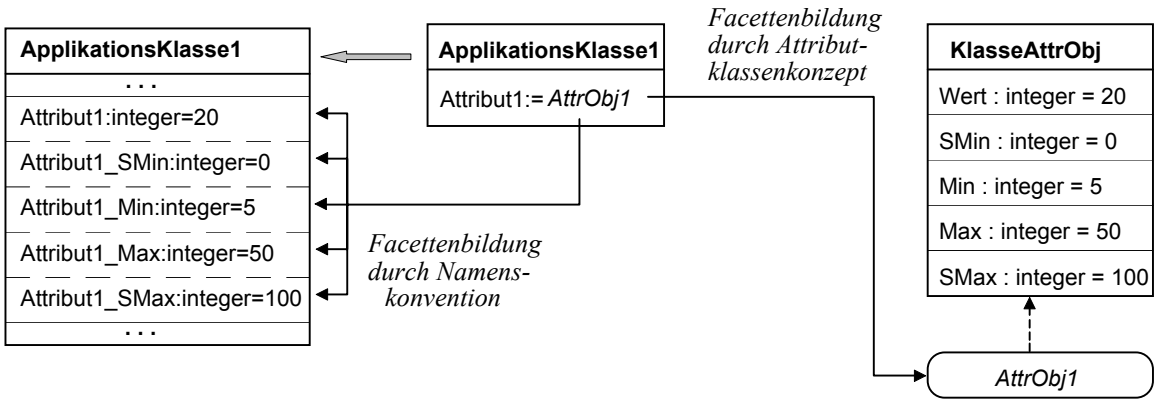


Abb. 7-35 Implementation eines Facettenkonzepts

In einem ersten Prototyp war sowohl der Wissens- als auch der Projektmodul in KappaPC implementiert. Der Projektmodul wurde wegen der angestrebten grafischen Arbeitsweise jedoch später in AutoCAD reimplementiert. Dort wurde als eine Variante auf der Basis Windows DDE der Wissens- und der Projekteditor so implementiert, daß sie parallel nutzbar waren. Die gesamte Objektverwaltung leistet KappaPC, in AutoCAD erfolgte lediglich die Visualisierung der Planungsobjekte (Instanzen). Dieses Vorgehen hat verschiedene Vorteile, so hat etwa die Änderung des Klassenschemas sofort Auswirkungen auf das Projektmodell. Ein weiterer Vorteil ist, daß die Methoden zur Plausibilitätssicherung und zur persistenten Speicherung von beiden Modulen genutzt werden können. Wegen der mangelhaften Laufzeit-effizienz und -stabilität wurden beide Module im aktuellen Prototyp jedoch entkoppelt (siehe [KOL94]), die Kommunikation erfolgt gegenwärtig über Dateien der o.g. Formate. Ein weiterer Grund hierfür war, daß die Klassenbasis eine im Vergleich zur Projektverwaltung niedrigere zeitliche Veränderlichkeit besitzt. Die zugehörigen Klasseneditoren werden also wesentlich seltener benötigt. Unter AutoCAD mußte deshalb eine eigenständige Objektverwaltung implementiert werden, die mit dem AutoLisp-Objektsystem *ALOS* vorliegt (siehe Anhang D). Auch hier wurde die Facettenverwaltung über eine Namenskonvention simuliert. Damit ergibt sich die aktuelle Systemarchitektur entprechend Abb. 7-36.

Bei der Konzeption des Nutzerinterfaces wurde nach folgenden zwei Strategien verfahren, wie sie im APPLE-Styleguide [APP90] formuliert sind:

- *Modellbezogenheit*

Das Ergebnis und die Verfügbarkeit von Aktionen orientiert sich am Modell, nicht am Programmzustand – gleiche Aktionen liefern gleiche Ergebnisse. In konventionellen Progammen wird jedoch häufig zwischen verschiedenen Programmodi unterschieden. Dies ist wenig nutzerfreundlich, insbesondere wenn Nutzer unbeabsichtigt in einen solchen Modus geraten bzw. sich des jeweiligen Modus nicht bewußt sind.

- *Ereignisbezogenheit*

Der zentrale Steuermechanismus interaktiver Programme ist eine ‘Event-Loop’, der auch für die FUNPLAN-Module implementiert wurde. Die Abfolge von Modellierungsaktionen (Events) wird nicht starr vorgegeben, sondern wird durch den Nutzer bestimmt.

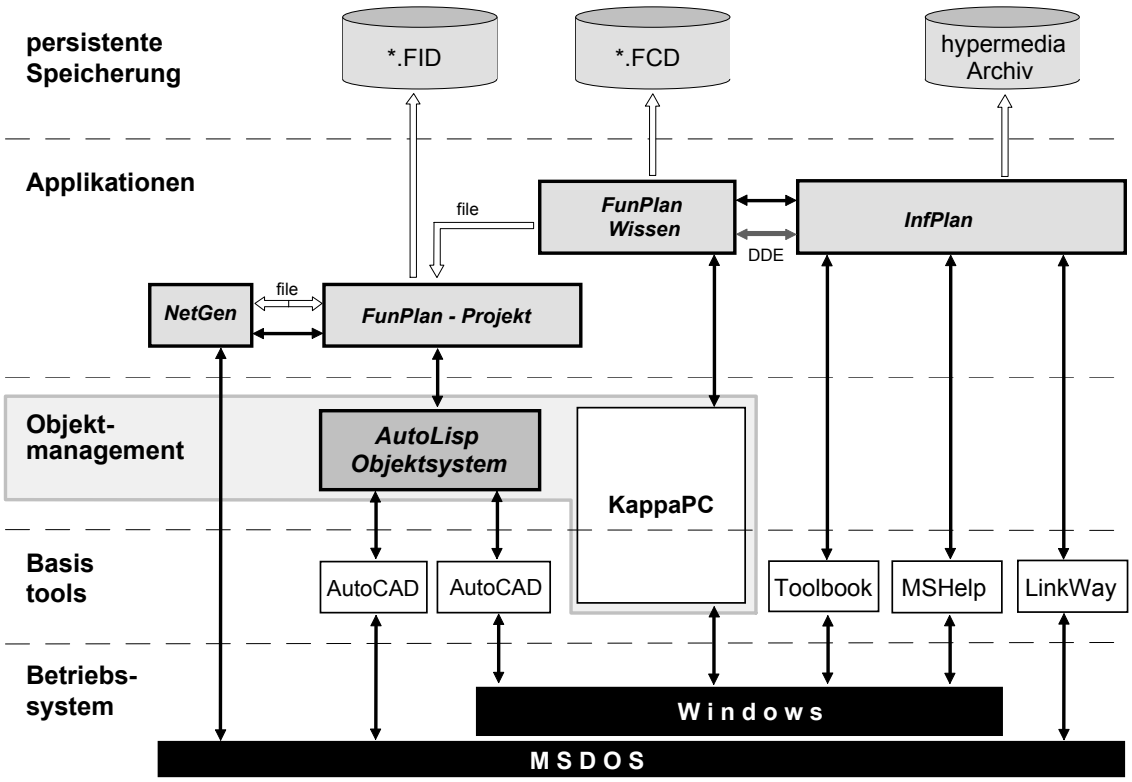


Abb. 7-36 FUNPLAN - Systemarchitektur

Als Interaktionsstil wird die objektorientierte Arbeitsweise '*see-and-point*' genutzt [APP90]. Eine modellierende Aktion des Nutzers erfordert zunächst die Selektion eines Objekts und bezeichnet dann durch Menüauswahl die Aktion. Auf diese Weise kann nach Selektion der Operanden die Menge möglicher Operationen meist eingeschränkt werden. Der vereinfachte Zustands-Übergangs-Graph von FUNPLAN-Projekt zeigt Abb. 7-37, die in den jeweiligen Zuständen nicht verfügbaren Aktionen werden in den Menüs gesperrt (siehe Tab. 7-2).

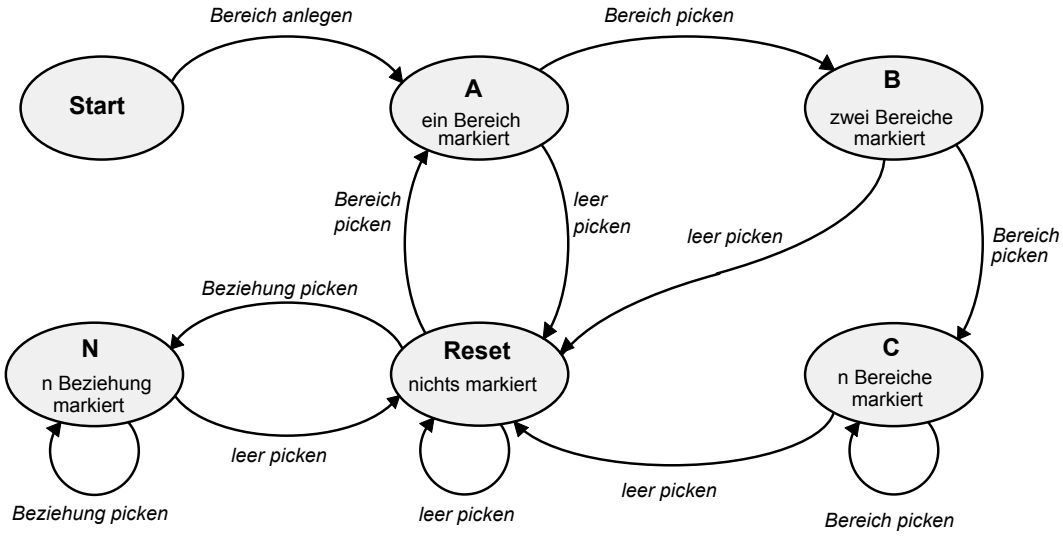


Abb. 7-37 Zustands-Übergangs-Graph für FUNPLAN - Projekt (nach [KOL94])

mögliche Aktion	Zustand					
	Start	Reset	A	B	C	N
Wissen laden	■	■	–	–	–	–
Wissen anzeigen	■	■	–	–	–	–
Projekt neu	–	■	–	–	–	–
Projekt laden	■	■	–	–	–	–
Projekt sichern	–	■	–	–	–	–
Bereich anlegen	■	■	–	–	–	–
Bereich beschreiben	–	–	■	–	–	–
Bereich verschieben	–	–	■	■	■	–
Bereich skalieren	–	–	■	–	–	–
Bereich löschen	–	–	■	■	■	–
Beziehung anlegen	–	–	–	■	–	–
Beziehung löschen	–	–	–	–	–	■
Undo	–	■	–	–	–	–

Tab. 7-2 Zuordnung verfügbarer Befehle zu Zuständen ([KOL94])

Um eine im Sinne des Entwurfsprozeßmodells explorative Arbeitsweise zu gewährleisten, sollten alle Aktionen rücknehmbar sein, kritische Aktionen sollten durch zusätzliche Bestätigungen abgesichert werden (‘*forgiveness*’ im Sinne von [APP90]). Ein UNDO-Mechanismus ist im ersten Prototyp implementiert worden. Wegen des hohen Aufwands wurde in späteren Varianten auf ihn verzichtet [KOL94]. Die in [APP90] geforderten Eigenschaften wie *‘metaphores from the real world’* und *‘direct manipulation’* sind durch die streng objektorientierte Arbeitsweise a priori gegeben. Die Forderungen ‘*consistency’* bzw. *‘plausibility’* wurden im Rahmen des Constraintkonzeptes implementiert.

Die Implementation des INFPLAN-Modul wurde bereits erläutert, in TOOLBOOK erfolgte sie in der entsprechenden Scriptsprache. Für die LINKWAY- und WindowsHelp-Prototypen wurden die jeweiligen externen Programe aufgerufen.

Die persistente Datenhaltung beruht gegenwärtig auf ASCII-Files. Eine Variante zur Datenhaltung und zum Datenaustausch auf der Basis einer relationalen Datenbank (PARADOX) wurde experimentiert, aber wegen mangelnder Effizienz verworfen ([KAI94]).

Gegenwärtig wird an einem Interface gearbeitet, um sowohl Projektmodelle als auch Klassenbasen im standardisierten STEP-Format schreiben zu können. Es basiert auf der ebenfalls in Bearbeitung befindlichen neutralen Klassen- und Instanzschnittstelle ‘AKO’, die die jeweiligen Metaobjektsysteme zu Verfügung stellen werden. ([KOL97]).

Bei der Nutzung zu Ausbildungszwecken wurde der Projektmodul bis zu folgender Modellgröße getestet: 60 Nutzungseinheiten aus 40 Klassen, mit jeweils ca. 3 Relationen und 8 Attributen. Das dort bereits zu verzeichnende mangelhafte Laufzeitverhalten folgt aus dem Overhead an Zeichnungsfunktionalität von AutoCAD sowie der Layerstruktur, die zur Sichtbarkeitssteuerung verwandt wurde.

Für einen allgemeinen Einsatz des vorgestellten Programmes ist die Abhängigkeit von AutoCAD hinderlich. Für große Modelle ist besonders unter den Aspekten Laufzeit und Ressourcenverbrauch eine Neuimplementation der Objektverwaltung unerläßlich. Hierzu wurde das Modellverwaltungssystem FLEXOB in C++ konzipiert und implementiert.

7.6 Bewertung der Prototypen

Im Vergleich zu heute üblichen CAD-Systemen, die i.allg. nur eine Automatisierung der Zeichenarbeit anstreben, verfügt FUNPLAN im Rahmen des vorgestellten Systemkonzeptes über

- eine frei benutz- und konfigurierbare Überprüfungs- und Beratungskomponete, die Wissen über Entwurfsobjekte in Form eines Klassensystems nutzt
- die integrierte Verwaltung *aller* sachlich zusammenhängenden Informationen in Modellobjekten eines Projektmodells
- die individuelle Systemerweiterbarkeit und -anpaßbarkeit durch die Möglichkeit zur Wissensrevision und die Toolarchitektur im PREPLAN-Systemkonzept.

Mit dem Grundsatz, Gebäudemodelle statt (zeichnerische) Repräsentationen zu schaffen, wird in PREPLAN ein Ziel verfolgt, das über die Unterstützung nur früher Phasen hinaus reicht: die integrierte Gebäudeplanung und -verwaltung. Es werden im Rahmen eines modell-/objektbasierten CAD-Konzeptes jene Datenstrukturen geschaffen, die über alle Lebensphasen des Bauwerks hinweg seine Abbilder in Form rechnerinterner Modelle prägen.

Mit den ersten implementierten Tools werden vordergründig die Aspekte Funktion und zuzuordnender Raum betrachtet. Die räumliche Gliederung des Bauwerks findet sich in der Aufbaustruktur wieder. Diese Gebäudegliederung tritt während des Entwerfens immer wieder auf, beispielsweise in Form eines Raumbuches oder später bei der Gebäudenutzung, wenn eine computergestützte Gebäudeverwaltung (Facility-Management) vorgesehen ist. Die Aufbaustruktur bildet den Kern der Modellstruktur über alle Entwurfsphasen hinweg. Sie ordnet Raum- bzw.- Funktionenobjekte, welche über den gesamten Bauwerkslebenszyklus erhalten bleiben. Damit werden bereits in den frühen Entwurfsphasen die Voraussetzungen für ein durchgängiges, multiple auswertbares Objektmodell geschaffen. Durch die Verfügbarkeit des Kontextes von funktionalen Modellen (in Form der Klassenbasis aus der sie erzeugt wurden) sind Modelltransformationen mit geringerem Aufwand, vor allem aber mit einem geringeren Informationsdefizit leistbar. Die Modelle selbst beschreiben das Bauwerk in einer syntaktisch und semantisch adäquaten Weise, so daß eine Interpretation der Modelle auch weit nach dem Zeitpunkt ihrer Erstellung wesentlich erleichtert wird. Konkrete und abstrakte Entwurfselemente wie auch deren Merkmale und Namen stammen aus der Begriffswelt des Entwerfenden.
Die gegenwärtig verfügbaren Beschreibungselemente erlauben nur die Repräsentation deskriptiver Objektmerkmale bzw. -zusammenhänge durch den Nutzer. Prozesse, d.h. Objektverhalten sind momentan nicht abbildbar. Für die Beschreibung von Bauwerken stellt dies keinen besonderen Nachteil dar, solange keine Simulation geplant ist.

Für einen Praxiseinsatz ist fraglich, inwieweit Architekten gegenwärtig in der Lage bzw. bereit sind, ihr Wissen zu formalisieren und in computerauswertbaren Wissensbasen zu vergegenständlichen. Es ist noch nicht untersucht, welcher Menge des Entwurfswissens sich in Taxonomien und den beschriebenen einfachen Randbedingungen erfassen läßt und ob diese dann zu einer allgemein akzeptierten Hilfestellung führt. Erfahrungen mit Architekten

zeigen, daß es schwierig zu erklären ist, warum zunächst Wissen erfaßt und geordnet werden muß. Die Vorgehensweise widerspricht der verbreiteten Meinung, daß die Bearbeitung von Projekten mit CAD vor allem schneller ist. Dieser Effekt des Geschwindigkeits-, jedoch vor allem des Qualitätsgewinns, ist erst mit konsolidierten Wissensbasen zu erwarten.

Das Systematisieren und Assistieren des Enwurfsprozesses durch CAD-Systeme nach dem Prinzip '*spezifiziere* ⇨ *realisiere* ⇨ *analysiere'* ermöglicht eine adäquate CA(A)D-Stützung. Es kann allerdings nicht verschwiegen werden, daß der Anfangsaufwand bei Nutzung solcher Systeme hoch ist. Dies ist begründet durch den Aufwand, der entsteht bei der

- *Wissenserhebung*

Dieser Vorgang ist bei konventionellen, nichtanpaßbaren Systemen zum Zeitpunkt des Kaufes bereits abgeschlossen, er wird also durch den Programmierer im Vorfeld geleistet. Bei den Modulen von PREPLAN wird eine gewisse Menge von anerkanntem 'Standard'-Wissen in Form von Objektklassen mitgeliefert. Die Leistungsfähigkeit, aber vor allem die Adäquatheit, steigt jedoch mit der Menge des eingebrachten persönlichen Wissens.

- *Modellierung*

Im Gegensatz zum Erstellen von Zeichnungen sind zusätzliche Informationen einzugeben, die im Rahmen der aktuellen Repräsentation keine Rolle spielen. Erst in der Abfolge von Entwurfsphasen und der Behandlung verschiedener Entwurfsaspekte kommt es dann zu einer Reduktion des Aufwandes durch die Vermeidung redundanter Eingaben, aber vor allem durch eine Sicherung von Plausibilität und Konsistenz der Modelle. Die aktuelle Form der Honorarordnung ([HOAI]) bietet allerdings für den Architekten wenig Anreiz, solche Modelldaten zu erheben, die die Arbeit nachfolgender Ingenieurgewerke erleichtert.

- *Einarbeitung*

Die neue Funktionalität der beschriebenen CAD-Systeme führt zu einer neuen Sicht auf den architektonischen Entwurf und einem neuen systematischeren Arbeitsstil. Es muß allerdings die Bereitschaft zu dieser Arbeitsweise vorliegen. Die neue Funktionalität der Systeme ist zusätzlich zu erlernen, sie ersetzt herkömmliche Techniken, z.B. in Zeichnungseditoren **nicht,** sondern ergänzt sie.

Grundsätzlich ist die computergestützte *Entwurfstechnologie,* die FUNPLAN demonstriert, auf Grund des (scheinbaren) Mehraufwand vorerst für folgende Fälle angebracht:

- *Große Projekte*

Die Komplexität ist hier bedingt durch die Anzahl der Entwurfselemente und deren dichter Verknüpfung sehr hoch. Eine systematische Arbeitsweise und Computerstützung sind dann zwingend. Für spätere Versionen wird deshalb die Steigerung der Laufzeiteffizienz der Module, vor allem der Datenhaltung, angestrebt.

- *Stark arbeitsteilig bearbeitete Projekte*

Hier wird die Konsistenzsicherung zum Problem. Aufwand und Fehleranfälligkeit bei der ständigen Interpretation der Arbeitsergebnisse anderer am Entwurf Beteiligter steigt stark mit der Gruppengröße. Die Festlegung eines Bauwerksmodells oder wenigstens einer Systematik, in der dieses Modell erfaßt wird sowie die größere Adäquatheit der Modelle bringen hier Verbesserung. Teilmodelle können leichter in ein gemeinsames Modell überführt werden. Dies ersetzt *nicht* Verfahren zur Organisation arbeitsteiligen Entwerfens mit *einem gemeinsamen* Bauwerksmodell.

- *Projekte von 'seltenen' Bauwerkstypen*

Für seltene oder hochtechnisierte Bauwerkstypen (z.B. Krematorien, Krankenhäuser) hat das Planungsmittel Funktionsplan schon immer große Bedeutung. Das „sich Informieren" zur Erarbeitung der Entwurfsgrundlagen nimmt breiten Raum ein. Die Zahl der Verknüpfungen und Randbedingungen in so hochtechnisierten Bauwerkstypen, wie Krankenhäusern[14] ist groß, eine Computerstützung also sinnvoll. Entsprechendes Entwurfswissen ist rar. Wenn es gelingt, solches Wissen in entsprechend formaler oder informaler Form in Wissensbasen zu vergegenständlichen, steigt bei der Nutzung entsprechender Systeme die Entwurfsqualität.

Der verfügbare Prototyp von FUNPLAN läßt sich unter den genannten Einschränkungen vorerst für folgende Anwendunsgebiete sinnvoll einsetzen:

- *zu Lehrzwecken*

Das Vermitteln von Entwurfssystematik nimmt im Gegensatz zum Erlernen von Fähigkeiten der grafischen Repräsentation in der Ausbildung nur geringen Raum ein. Funktionspläne, ihre Semantik, ihre Einordnung und Nutzung im Entwurfsprozeß sind i.allg. im Studium kein Gegenstand expliziter Betrachtung. Der Modelliergedanke, im Sinne einer umfassenden Verwaltung und Strukturierung aller relevanten Bauwerksinformation, erfährt unter Architekten nur geringe Beachtung, wird aber zukünftig breiten Raum gewinnen. Der Vermittlung dieser Ideen bietet FUNPLAN eine technische Unterstützung.

- *zur Bestandsdatenerfassung*

Wenn der Vorrat an Funktionselementtypen (Taxonomie) für ein Bauwerk vollständig erfaßt wurde, kann mit ihrer Hilfe eine einfache und schnelle Bestandserfassung realisiert werden. Die erhobenen Daten können in der angebotenen Granularität bereits als Basis einer computergestützten Gebäudebewirtschaftung dienen.

- *als grafischer Editor für Funktionspläne*

Auch ohne die Benutzung der Möglichkeiten der Modellbildungen und der Konsistenzprüfung bleibt FUNPLAN ein Mittel zur rein grafischen Erstellung von Funktionsplänen, bis hin zu deren Druck. Hierzu wird keine ausgefeilte Wissensbasis benötigt. Besonders der Modul NETGEN zur Entflechtung und übersichtlichen Darstellung von Funktionsgraphen ist hier von Wert.

Einen praktischen Einsatz und Erprobung erfuhr FUNPLAN auch

- *zur prototypischen Entwicklung von Domänenmodellen*

u.a. im Projekt 'GEBIS'[15] (Gebäudeinformationssystem, 'schnelles Aufmaß') und im Projekt 'Ökologisches Bauen'[16] (Energetische Langzeitbewertung von Bauwerken, Ökologisches Raumbuch).

[14] siehe z.B. DIN 13080 „Gliederung des Krankenhauses in Funktionsbereiche und Funktionsstellen"
[15] Prof. Donath/ Bauhaus-Universität Weimar
[16] Prof. Gronau/ Bauhaus-Universität Weimar

8. MODELLVERWALTUNGSSYSTEME – IMPLEMENTIERUNGSTOOLS FÜR DYNAMISCHE CAD-SYSTEME

8.1 Anforderungen

Zielrichtung der verfügbaren Entwicklungstechnologien und damit der entsprechenden Tools ist eine effiziente Softwareproduktion, und hierbei besonderes kurze Entwicklungszyklen. Selbst Systeme, die prinzipiell eine dynamische Veränderung von Domänenmodellen ermöglichen (PLAKON [CUN91] oder CEMENT [BER95]), gehen von einer klar getrennten Sequenz

Modellentwicklung ⇨ *Modellvergegenständlichung* ⇨ *Modellinstanziierung/Nutzung*
OO-Problemanalyse ⇨ OO-Softwaredesign ⇨ (z.B. OO-CAD-Systeme)

aus. Jede Modellnutzung impliziert einen mittelbaren Kommunikationsprozeß zwischen Entwickler und Nutzer über das Medium Software, da der Nutzer das implementierte Domänenmodell kennen bzw. interpretieren muß. Da Kommunikation im Sinne von WINOGRAD ([WIN86]) immer ein verlustbehafteter Prozeß ist, stellt sich durch die rückkopplungsfreie Entwicklungstechnologie letztlich immer die Frage nach der Adäquatheit von Software.

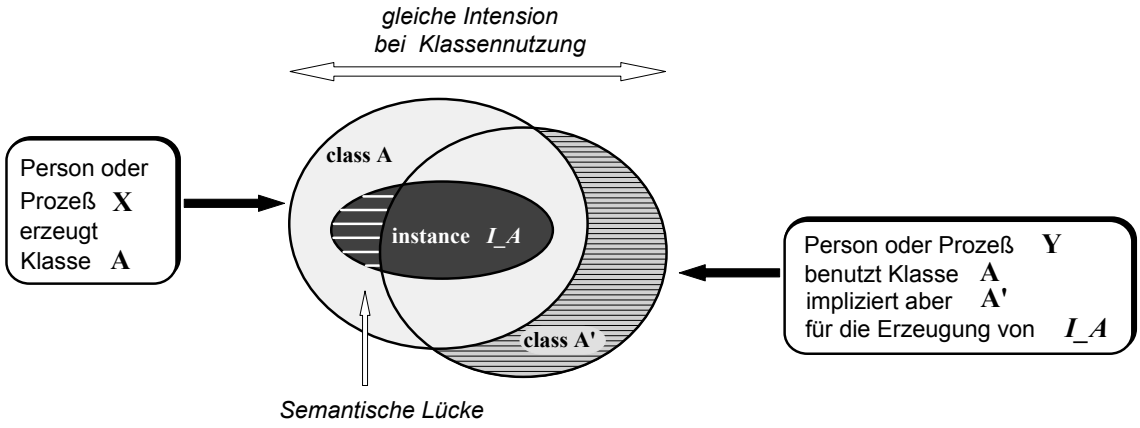


Abb. 8-1 Semantische Lücke beim Erzeugen und Nutzen einer Klasse

Eine Möglichkeit der Rückwirkung auf die Phase des Softwareerstellungsprozesses, die der Nutzung vorgelagert ist, ist es, entsprechend versierten Nutzern eine Veränderung des Domänenmodells in zumindest dessen deskriptiven Teilen zu ermöglichen. Voraussetzung hierfür ist die explizite Verfügbarkeit des Domänenmodells und eben die Möglichkeit seiner Veränderung zur Laufzeit. Besonders für CAD-Prozesse, die wie der architektonische Vorentwurf von hoher Dynamik und Subjektivität geprägt sind, stellt diese Forderung eine Voraussetzung für adäquate Software dar.

Zentrale Aufgabe eines entsprechenden Modellverwaltungssystems (MVS, siehe Glossar) soll neben der

- *Verwaltung von Modelldaten (Projekte)* auch die Unterstützung der *Domänenmodellentwicklung und -adaption* durch Beschreibung eines relevanten Weltausschnittes

mit den Mitteln des objektorientierten Paradigmas sein.

Ein solcher Modellierkern muß Funktionen bereitstellen zur rechnerinternen Repräsentation:

- der Entitäten dieses Weltausschnittes
- ihrer Eigenschaften und ihres Verhaltens
- ihren Beziehungen zu anderen Entitäten
- ihrer Aufbaustruktur, soweit es sich um konkrete Entitäten handelt.

Stellen diese Forderungen an sich noch keine Novität dar, da sie Bestandteil des objektorientierten Paradigmas selbst sind, werfen sie doch für den Softwareentwicklungsprozeß eine Reihe von Fragen auf. So stellt HEINECKE fest:

> "Zu einem echten Werkzeug der Konstrukteure können objektorientierte (Tools) erst dann werden, wenn diese die Möglichkeit erhalten, neue Klassen und Objekte interaktiv zu erzeugen... Wie solche Möglichkeiten realisiert werden können, ohne daß die Konstrukteure vertiefte Programmierkenntnisse benötigen, ist dabei noch weitgehend offen." [HEI94]

Im weiteren wird für die Bildung von Domänenmodellen[1] von der Annahme ausgegangen, daß es nicht möglich ist, in der Phase des Softwaredesigns ein allumfassendes Modell zu erstellen, welches statische Gültigkeit erlangt. Diese Feststellung hat weitgehende Konsequenzen für das zugrundeliegende Modellierwerkzeug. Im Vordergrund des zu entwickelnden Modellierkerns stehen primär Aufgaben der Objektverwaltung, weniger die maschinelle Dateninterpretation. Sie wird in spezifischen Applikationsprogrammen zu lösen sein, wie FUNPLAN eines darstellt. Die in der Künstlichen Intelligenz dominierenden interpretierenden Programmiersysteme vermögen zwar die im weiteren beschriebenen Anforderungen (Tab. 8-1) zu erfüllen, sind jedoch vorwiegend für den Bau von Expertensystemen ausgelegt und verfügen zusätzlich zur Objektverwaltung über eine Reihe weiterer Komponenten (Inferenzmaschinen, Debugingtools etc.). Für bloße Verwaltungszwecke sind sie zu mächtig und daher ineffizient (auch kostenineffizient). Ziel ist es, ein kleines, im Funktionsumfang auf das Anforderungsspektrum passend zugeschnittenes, flexibles Objektverwaltungsmodul zu schaffen, das die geforderten Leistungsmerkmale bei möglichst hoher Laufzeiteffizienz erreicht, so daß auch realistische komplexe Bauwerksmodelle erstellt und bearbeitet werden können.

In Taballe 8-1 werden Funktionen aufgeführt, die als eine orthogonale Menge von Basisoperationen durch ein dynamisches MVS zu realisieren sind. Um bei Anwendungsentwicklern als potentielle Nutzer auf Akzeptanz zu stoßen, sollte das Modellierwerkzeug aufbauend auf der Basismenge von Kernfunktionen über die folgenden weiteren Eigenschaften verfügen:

- Analyse der Domänenstruktur (Analyse der Struktur der Taxonomie sowie von Klassen- und Instanzobjekten)
- Analyse der Projektstruktur (Aggregationsanalyse)
- domänenspezifische Attribute und Relationen mit anwenderdefinierter Semantik
- Abbildung von Unschärfe und Unvollständigkeit (in speziellen Slotobjektklassen)
- Integration informalen Wissens (multimediale Erweiterungen, siehe INFPLAN)
- Unterstützung der Konsistenzsicherung (Monitorfunktionen).

[1] Produktmodellentwicklung im Rahmen eines konventionellen Entwicklungszyklus

BASISFUNKTIONEN FÜR OO-MODELLVERWALTUNGSSYSTEME		
Klassenobjekte	***Instanzobjekte***	***Attribut- und Relationenobjekte***
Objektfunktionen	Objektfunktionen	
`ErzeugeKlasse, ErzeugeInstanz` (K)		
`LöscheKlasse` (D)	`LöscheInstanz` (D)	
`ErzeugeAttribut, ErzeugeRelation` (K)		
`LöscheAttribut, LöscheRelation` (D)		
`HoleSuperKlassen, HoleSubklassen, HoleInstanzen` (I)	`HoleKlassen` (I)	
	Aggregatfunktionen	
	`BaueEin, BaueAus` (Z)	
	`HoleKomplexe, HoleTeile` (I)	
Slotfunktionen		Facettenfunktionen
`HoleAttribut, HoleRelationen` (Z) (realisieren Werte- und Struktur-Erbschaft)		`HoleFacette, SetzeFacette` (Z) (realisieren Werte-Erbschaft)
`HoleAttributListe, HoleRelationenListe` (I) (realisieren Struktur-Erbschaft)		`HoleFacettenListe` (I)

(K) *Konstruktionsfkt.* (D) *Destruktionsfkt.* (Z) *Zugriffsfkt.* (I) *Informationsfkt.*

Tab. 8-1 Basisfunktionen zur dynamischen Objektmodellverwaltung

Insbesondere die Analysefunktionen lassen sich ohne großen Programmieraufwand durch die Basisfunktionen darstellen. Da ein solches System als Modellierungskern in CAD-Systemen Verwendung finden soll, wird eine explizite Unterstützung der ‘has-part’/’part-of’ Relation vorgeschlagen. Auch hier lassen sich mit den angeführten Funktionen alle höheren Funktionen zum Aufbau und zur Analyse komplexer Aggregationsgraphen darstellen.

Um den Aufwand der Neuimplementation eines solchen Systems zu begründen, zeigt Tabelle 8-2 eine empirische Bewertung einiger ausgewählter Systeme verschiedener Toolkategorien hinsichtlich der Erfüllung charakteristischer Anforderungen an eine dynamische Basisobjekt-verwaltung. Ergebnis ist, daß keines der untersuchten Systeme alle Anforderungen in optimaler Weise erfüllt. Die Bewertung basiert auf den jeweiligen Handbüchern der Systeme. Die Bewertungskriterien resultieren aus den angeführten Anforderungen an ein MVS sowie aus einer Untersuchung zu hybriden Expertensystemtools in [SCH93].

Anforderungen	C++	ONTOS	KappaPC	FlexOB	ALOS
Portabel und verfügbar	■	–	–	☐	☐
Genügend laufzeiteffizient	■	☐	–	☐	–
Genügend kosteneffizient (Preis)	■	–	☐		
Genügend entwicklungseffizient (Eignung zum Prototyping)	–	–	■	–	■
Laufzeitdynamische Objektstrukturen, Löschen, Erzeugen von Klassenobjekten	–	☐	■	■	■
Struktur- und Wertvererbung	–	☐	■	■	■
Umklassifizieren von Instanzen/Subklassen	–	–	☐	■	■
Löschen, Erzeugen von Attributen/Relationen	–	■	■	■	■
Unterstützung von Slotfacetten	–	–	☐	■	–
Unterstützung von Relationenobjekten	–	–	–	■	–
Unterstützung der Kompositionshierarchie	–	☐	–	■	■
Weitgehende Objekt-, Taxonomie- und Aggregationsanalyse	–	☐	■	■	■

■ erfüllt ☐ bedingt erfüllt – nicht erfüllt

Tab. 8-2 Analyse verschiedener Systeme zur Entwicklung von OO-Anwendungssystemen[2]

8.1.1 MetaObjekte

Unter der Maßgabe, daß ein solcher Modellierkern selbst wiederum objektorientiert erstellt wird, muß er Metaobjektklassen für folgende paradigmatische Objekttypen bereitstellen:

MetaObjekte	
• *ObjektObjekte*	• *SlotObjekte*
Klassenobjekte	Attributobjekte
Instanzobjekte	Relationenobjekte

Für Klassenobjekte und Instanzobjekte existiert eine weitgehende strukturelle Konformität. Sie sind beide Objekte im Sinne der Kapselung. Ihr Aufbau aus Slots sowie der Algorithmus der Vererbung von Slots sind gleich. So erscheint es sinnvoll, beide Objekttypen in einer Metaklasse ‘ObjektObjekte’ zusammenzufassen und gemeinsam zu implementieren.

Für die Gestaltung des Modellierkerns ist festzulegen, welche Bereiche des Domänenmodells statisch, d.h. zur Laufzeit unveränderlich, und welche dynamisch änderbar sind. Aus programmiertechnischen sowie Effizienzgründen gehören Methoden zum statischen Teil des Domänenmodells. Hieraus folgt, daß Modellklassen als Instanzen des Typs ‘ObjektObjekt’

[2] Es handelt sich hier um eine willkürliche Auswahl. Die getroffen generalisierten Aussagen müssen für spezielle Projekte jeweils geprüft werden. FLEXOB und ALOS werden im weiteren vorgestellt.

diese Methoden kapseln müssen. Die Ableitung von Domänenklassen sowie deren Attributierung sollen dagegen dynamisch sein.
Der Nutzer bildet ausgehend von den statischen, wurzelnahen Klassen sein begriffliches Wissen in Taxonomien ab. Damit lassen sich in eleganter Art und Weise viele der in der Arbeit erhobenen Ansprüche erfüllen:

- Terminologische Strukturierung des Domänenwissens,
  inklusive des informalen Zusatzwissens
- Nutzung der Begriffswelt des Anwenders
- Standardannahmen durch Vererbung,
  sowohl auf (Sub-)Klassen- als auch auf Instanzenebene
- Modellierung von Abstraktionsunschärfe
  durch dynamische Spezialisierung von Instanzobjekten
- Inkrementelle Erweiterbarkeit des Modells
- Gute grafische Repräsentierbarkeit des Taxonomiengraphen.

Für die strukturellen Aspekte von Objekten (Attributierung) sowie deren Wertbelegung ist dem Nutzer eine dynamische Änderbarkeit bzw. Ableitbarkeit (Subklassenbildung) zur Laufzeit zumutbar, ja sogar wünschenswert. Grundsätzlich ist darüber hinaus eine dynamische Erweiterung des Vorrates an Slotobjektklassen (Typen verfügbarer Attribut- bzw. Relationenarten) denkbar. Es scheint jedoch fraglich, ob die entstehende Komplexität durch Programmierer, doch vor allem vom Anwender beherrschbar ist und ob sich hierfür effiziente Implementationen finden lassen.

Die getroffenen Prämissen führen zu einer *semidynamischen Arbeitsweise*. Alle Klassen, die über spezielle Methoden verfügen sollen oder die sich in ihrer Semantik von ihrer Superklasse unterscheiden, müssen zum Entwicklungszeitpunkt statisch, d.h. unabänderlich vorimplementiert werden. Dasselbe gilt für Struktur und Funktionalität der anzubietenden Slotobjektklassen.

8.1.2 Attribut- und Relationenobjekte

Attribute und Relationen bestehen aus einer Menge Facetten und zugeordneten Prozeduren (Zugriffsmethoden, allgemeine Methoden, prozedurale Erweiterungen). Diese Vorgehensweise hat gegenüber atomaren Attributen (terminale Datentypen der zugrundeliegenden Wirtssprache) und Relationen ((Adreß-) Referenzen) mehrere Vorteile ([WEH95]):

- Durch Facettierung lassen sich domänenspezifische Zusatzinformationen ablegen (Maßeinheiten, Wertemengendeskriptoren, Kommentare, ...)
- Modellierung von attributiver und struktureller (relationaler) Unschärfe ist möglich
- Die Zugriffsmethoden können Prozeduren zur Konsistenzsicherung, Plausibilitätskontrollen, Zugriffsüberwachung sowie ggf. zur Regelpropagation triggern
- Die gesamte Funktionalität von Attributen und Relationen und damit deren Anwendungssemantik ist in Objekten gekapselt; Klassen- und Instanzobjekte verwalten nur neutrale Referenzen (Referenzen auf MetaObjekte)
- Angepaßte, generische Dialogmasken für die alphanumerische Ein- und Ausgabe sind einfach implementierbar
- Modellierung erweiterter Relationentypen, wie gerichtete, ungerichtete, bewertete Relationen, ist realisierbar

- Inverse Relationen für antisymmetrische Relationen könne automatische generiert werden
- Die Implementierung von Containern für nichtalphanumerische Attribute, die auch informales Wissen repräsentieren können (Bilder, Klänge, Texte, ...), ist möglich.

Daraus resultiert eine deutliche Erhöhung der Modelliermächtigkeit gegenüber der Nutzung atomarer Attribute.

Obwohl es prinzipiell möglich wäre, alle vorkommenden Relationen durch Relationenobjekte zu modellieren, erscheint dies für die paradigmatischen Relationen (siehe Glossar)

- Generalisierung / Spezialisierung
- Klassifizierung / Instanziierung
- Aggregation / Zerlegung

nicht sinnvoll. Da diese Relationen zur Organisation des Objektsystemes extensiv genutzt werden, sollten sie aus Effizienzgründen als echte Membervariablen der Objektobjekte realisiert werden. Für CAD-Anwendungen zählt hierzu besonders die Aggregationsrelation. Auch wenn diese Relation sowohl von ihrer Semantik als auch durch ihren heterarchischen Charakter für auf ihr beruhende Algorithmen problematisch ist, hat sie vor allem für die Navigation in komplexen Modellen sowie für die Organisation des Entwurfsprozesses eine große Bedeutung (siehe FUNPLAN).

8.1.3 Persistenz und Objektidentität

Neben den primären Aufgaben der Modellverwaltung muß der Kern auch Funktionen zur persistenten Speicherung bzw. Archivierung von Modell- und Projektdaten bieten. Dies ist einerseits sinnvoll, da der Kern selbst nur wenige generische Objekttypen kennt und in diesen die erforderliche Funktionalität gekapselt werden kann. Andererseits sind Funktionen zur Verwaltung von Projekt(daten-)banken bzw. die Implementation von Transaktionskonzepten und 'adhoc'-Abfragesprachen (OMQL[3]) stark vom Zielsystem abhängig. Deren Implementation sollte in Schichten vorgenommen werden, die auf dem Kern aufbauen.
Wegen der angestrebten Langlebigkeit der Modelle macht es Sinn, diese in einer vom Menschen lesbaren Form zu speichern. Binäre Streams oder objektorientierte Datenbanken, wie etwa ONTOS, sind hierzu nicht geeignet. Schon ein Versionswechsel der Datenbankmaschine kann bedeuten, daß bestehende Daten nicht mehr verlustfrei übernommen werden können. Deshalb sollte die Archivierung in strukturierten Textfiles mindestens als eine Variante angeboten werden. Prinzipiell eignen sich hierzu textbasierte Standardaustauschformate wie etwa STEP, wobei sie allerdings den Austausch bzw. die Speicherung von Schemata nicht unterstützen. Gerade diese ist jedoch ein wesenliches Leistungsmerkmal des angestrebten Modellverwaltungstools.
Um die Interpretierbarkeit der Projektdaten zu erhöhen, sollten die eigentlichen deskriptiven Modelldaten von den lediglich implementationsrelevanten Zusatzinformationen (z.B. Farbcodes oder Bildschirmkoordinaten in Editoren) getrennt gespeichert sein.

In engem Zusammenhang mit der Persistenz steht die Problematik der Objektidentität. Sie dient zur eindeutigen Unterscheidbarkeit eines Objektes von allen anderen Objekten.

[3] Object Model Query Language, siehe [RAN92]

Nach HUGHES ([HUG91]) ist die Unterscheidbarkeit zu gewährleisten in Bezug auf:

- *Unabhängigkeit vom Ablageort (location independence)*
  Die Identität soll invariant gegenüber Änderungen des Speicherortes sein (Hauptspeicher oder Sekundärspeicher).
- *Unabhängigkeit vom Wert (value independence)*
  Veränderungen der Objektwerte dürfen nicht zu Änderungen der Objektidentität führen.
- *Unabhängigkeit von der Struktur (structure independence)*
  Die Objektidentität darf sich bei Variation der Objektstruktur nicht verändern.

Objektidentität kann hierzu auf verschiedene Art und Weise implementiert werden:

- *Identität durch Adressierung*: Jedes Objekt hat eine eindeutige Speicheradresse. Bei dieser Art ist der Objektidentifikator nicht Bestandteil des Objektes selbst, die Forderung nach *location independence* ist nicht erfüllt (Vorteil: schneller Zugriff).
- *Identität mit Schlüsseln:* Der Identifikator wird durch einen eindeutigen Schlüsselwert repräsentiert. *Location independene* wird mit dieser Form erreicht, nicht dagegen *value independence*, da der gewählte Schlüsselwert potentiell dem Nutzereinfluß unterliegt.
- *Identität mit Surrogaten*: Surrogate sind systemgenerierte, global[4] eindeutige Bezeichner, die unabhängig vom Speicherort sind. Jedem Objekt wird bei dessen Instanziierung ein Surrogatwert als Identifikator zugewiesen, der über dessen gesamte Lebensdauer unverändert bleibt (*starke Objektidentität*).

Bestimmte Leistungsmerkmale, wie die Trennung von Modell- und Implementationsdaten oder die Vereinigung mehrerer Projekte, lassen sich überhaupt erst sinnvoll über das Konzept der starken Objektidentität realisieren.

8.1.4 Propagation

Im allgemeinen existieren für Objekte in komplexen Modellen vielfältige Zusammenhänge verschiedener Typen. So haben Entwurfshandlungen oft sehr komplexe, weitreichende und tief geschachtelte Wirkungen, deren *Propagation* vom Nutzer nur selten überschaut wird. Zur Beschreibung solcher Auswirkungen sollten dem Applikationsentwickler, aber auch dem Nutzer geeignete Mittel zur Verfügung stehen. Diese Mittel unterscheiden sich jedoch in Expressivität, Einsatzzweck, Art der Propagation und damit in ihrer Effizienz beträchtlich. Tools zur Propagation sind deshalb nicht Bestandteil des Modellierkerns, sondern sie sind als Erweiterung zu diesem zu realisieren.

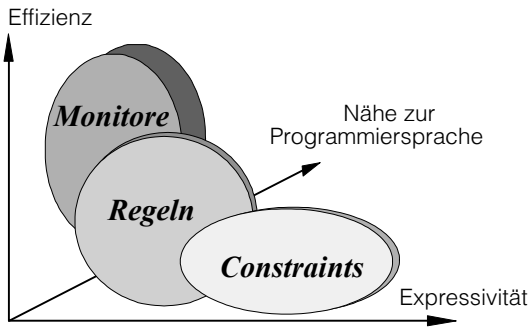


Abb. 8-2 Einordnung von Mitteln zur Propagation von Modellveränderungen

[4] tatsächlich im Sinne von weltweit

Nach Möglichkeit sollte die Beschreibung so formalisiert sein, daß spezifizierte Wirkungen automatisch ausgelöst und propagiert werden können. Die verschiedenen Mechanismen hierfür sind nicht Bestandteil der OO-Programmierung, sondern sind eher aus Programmierwerkzeugen der Künstlichen Intelligenz bekannt.

MONITORE[5]

Sie sind der effizienteste Weg zur Triggerung von Folgeaktionen, da hierzu nur der Zugriff selbst zu dem mit einem Monitor versehenen Slot nötig ist. In ähnlicher Weise läßt sich die Erzeugung oder Löschung von Objekten überwachen. Der Mechanismus selbst beachtet keine inhaltlichen Randbedingungen. Er ist damit Basis der Implementierung 'höherer' Sprachmittel. Da sie Funktionen des Systemskerns überwachen, können sie nicht in einer selbständigen Schicht ausgegliedert werden.
Durch die Homogenität des Slotbegriffes sind durch Monitore Prä- und Postprozesse abbildbar, wie sie z.B. aus OMT [RUM91] bekannt sind.

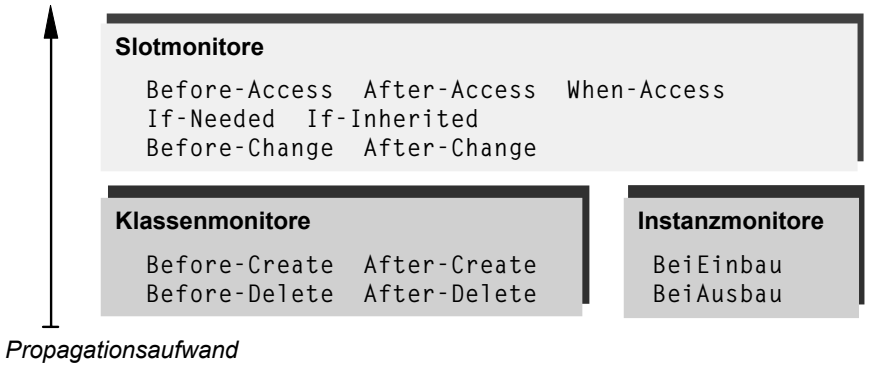


Abb. 8-3 Arten von Monitoren

Für CAD-Systementwicklung ist die paradigmatische Aggregationsrelation von großer Bedeutung. Die Veränderung dieser Relation, d.h. die Veränderung der kompositionellen Hierarchie, sollte durch eigene Monitortypen überwacht werden ('`BeiEinbau`', '`BeiAusbau`').

REGELN

Regeln verfügen als IF-THEN(-ELSE)-Konstrukte über eine Syntax, die natürlichsprachlichen Formulierungen näherkommt. Sie verfügen daher für die Domänenmodellierung über eine höhere Ausdruckskraft. Es gibt für Regeln verschiedene Standardauswertemechanismen, die im allgemeinen als Interpreter vorliegen. Eine Regelauswertung ist damit in jedem Fall ressourcenintensiver als Monitorüberwachung.

Der Einsatzzweck und die Auswertemethodik von Regeln korrespondieren mit dem Handlungstypus der unterstützten Entwurfshandlung, wie er sich aus deren Klassifikation ergibt (siehe Kap. 4):

Metaregeln gehören zum Handlungstypus 'control'. Sie steuern den Ableitungs- und damit den Entwurfsprozeß, indem sie nachfolgende Entwurfsaktionen in Abhängigkeit des Entwurfszustandes auswählen (siehe Kontrollstrategien in PLAKON [GÜN92]).

[5] derartige Techniken sind auch unter den Begriffen 'procedural attachments', 'Dämon' [PUP91], 'read-/ write-dependencies' (BABYLON) u.a. bekannt

Vorwärtsverkettende Regeln führen Modellveränderungen aus und gehören damit zum Handlungstypus 'Synthese' (Produktionsregeln).

Rückwärtsverkettende Regeln gehören zum Handlungstypus 'evaluation'. Sie ermitteln den Wahrheitsgehalt einer Hypothese (Ziel im Sinne von PROLOG). Der Ableitungsprozeß sollte seiteneffektfrei sein, d.h. es kommt im Zuge der Auswertung nicht zu Modellveränderungen. Das Bestimmen von Entwurfszielen (Hypothese) erfolgt durch Aktionen des Typs 'Analyse'.

CONSTRAINTS

Constraints stellen das ausdrucksstärkste Mittel zur Modellierung von Objekt- oder Prozeßabhängigkeiten dar. Sie verfügen im Gegensatz zu Regeln über keine bevorzugte Auswerterichtung, stellen also *ungerichtete* Relationen zwischen beliebigen Objekt-Slot-Paaren dar. Die Stelligkeit des Constraints folgt aus der Anzahl der einbezogenen Objekt-Slot-Paare. Constraints, bei denen diese Anzahl vorab unbekannt ist, sind mengenwertig (Beispiel: Die Fläche einer Etage (Einzelattribut) 'ist_gleich' (Constraint) der Summe der Flächen ihrer Räume (Menge von Attributen)).
Falls eine echte Wertepropagation angestrebt wird, erfordert dies einen komplexen und ressourcenintensiven Algorithmus. Entsprechende Algorithmen sind noch Gegenstand der Forschung[6]. Bei der Modellierung von großen sowie komplex strukturierten Modellwelten spielen sie in der Praxis gegenwärtig keine Rolle.

8.1.5 Systemarchitektur dynamischer modellbasierter CAD-Systeme

Die Anforderungen, die sich für die Unterstützung von CAD-Prozessen in dynamischen Modellwelten ergeben, haben einen anderen Systemaufbau sowie einen veränderten Entwicklungszyklus für solche Software zur Folge. Bei der Nutzung von Modellverwaltungstools werden die rein deskriptiven Modellanteile der Applikationen (Attribute und Relationen) von deren Funktionalität (Methoden und Prozeduren) getrennt. Heute wird die Implementation von Funktionalität jedoch von Softwareentwicklern geleistet, so daß eine Fixierung von deskriptiven Strukturen zum Entwicklungszeitpunkt unumgänglich ist. Im Rahmen des angestrebten Modellverwaltungskerns erfolgt diese Fixierung jedoch mit denselben Mitteln, wie für dynamische, d.h. später änderbare Modellanteile. Dies hat den Vorteil, daß das gesamte deskriptive Modell zugreifbar und eben partiell änderbar ist.
An Schnittstellen müssen nur semantikfreie Metaobjekte ausgetauscht werden, die Probleme domänenspezifischer Schnittstellenstandards entfallen. Werden etwa bei Verwendung von STEP Objekte transferiert, ist deren Typ, z.B. 'Raum', festgeschrieben. Den Kommunikanten müssen dabei Struktur *und* Semantik der ausgetauschten Objektinstanzen bekannt sein. Beim hier beschriebenen Vorgehen werden dagegen Objekte des Typs 'Object' ausgetauscht, deren Struktur (Aufbau aus Slotobjekten sowie paradigmatischen Relationen) ist durch das Metamodell genormt. Falls die Semantik der ausgetauschten Objekte unklar ist, besteht die Möglichkeit zur Domänenmodellanalyse, ist z.B. ein Objekt des Typs 'Küche' unbekannt, ist möglicherweise dessen Supertyp 'Raum' bekannt (Prinzip der Erbschaftspfade). Selbst wenn dem Zielsystem keine Superklassen des Ausgangssystems bekannt sind, wird ein Objekt des (Meta-)Typs 'ObjectObject' übertragen. Mit generischen Editoren, die sich nur auf das

[6] wie z.B. in CONSAT (GÜSGEN), BABYLON (VW-Gedas), PLAKON (Uni Hamburg), YACSS (Uni München), CHIP (HEUTENRYCK). Grundlegende Arbeiten stammen von G.J.SUSSMAN und G.L.STEELE (1975 - 1980) ([STE80]), eine umfassende Darstellung erfolgt bei STOYAN ([STO91]).

verwendete Metamodell stützen, kann ein solches Objekt immer noch betrachtet bzw. bearbeitet werden. Das Vorgehen ist nicht gänzlich neu. Der Erfolg von generischen Systemen, wie EXCEL (Metamodell 'Tabelle'), zeigt die Praktikabilität des Ansatzes.

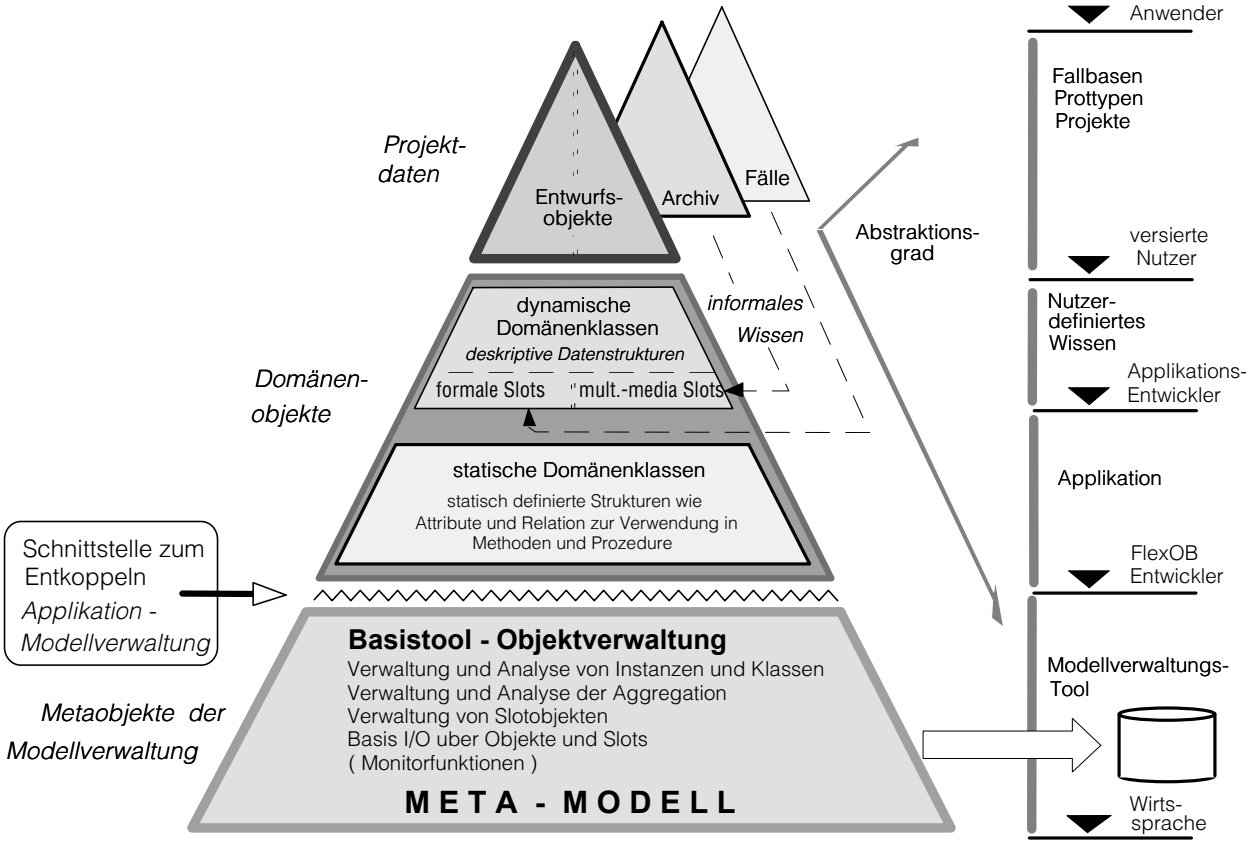


Abb. 8-4 Architektur dynamischer CAD-Systeme

Der Einsatz von Diensten, wie Transaktionenmanagement oder 'notification services' in einem CAD-Gesamtprozeß auf der Basis heterogener Tools, setzt ebenfalls ein gemeinsames Metamodell voraus, um die genannten Dienste zu implementieren.

Ein weiterer Vorteil ist, daß Modelladaptionen von *versierten Nutzern* am Ort und zur Zeit der Programmnutzung ausgeführt werden können. Dieses Vorgehen ist aus der Künstlichen Intelligenz seit langem bekannt. Im Rahmen von Expertensystementwicklungen nimmt diese hybride Stellung zwischen Nutzer und Entwickler ein '*knowledge engineer*' ein ([KAR90]). Dies scheint bei der gegenwärtig beobachteten rasanten Updatefrequenz von CAD-Software ein Weg aus der Softwarekrise.

8.2 ALOS – AutoLisp-Objektsystem

Objektorientierte Denk- und Arbeitsweisen werden in LISP schon lange gepflegt. Kommerzielle CommonLisp-Systeme (z.B. Allegro-CommonLisp) beinhalten in ihrem Lieferumfang das standardisierte Objektsystem *CLOS*[7] [KEE89]. Im Rahmen des bevorzugten Einsatzes von LISP für Zwecke der Künstlichen Intelligenz wurde mit der Frametechnologie eine objektorientierte Wissensrepräsentation verfügbar, die in einigen Punkten die Mächtigkeit heutiger objektorientierter Programmiersprachen übertrifft (z.B. Facetten). Es ist vor allem die Gleichbehandlung von Daten und Programmen (syntaktisch wie speichertechnisch), die LISP für solche Aufgaben prädestiniert und eine Metaprogrammierung ermöglicht.

Das CLOS-System als Teil des Sprachstandards CommonLisp ist ein effizientes Werkzeug zur objektorientierten Programmierung, alle entsprechenden Funktionen sind 'built-in', also maschinennah effizient programmiert und erweitern z.T. die Syntax der Sprache. Die Standardisierung führt zu portablen, gut lesbaren Programmen. Diese Implementation bringt aber auch Nachteile. Es ist nicht möglich, die Funktionalität des Objektsystems im Kern zu ändern oder zu erweitern um beispielsweise Komponenten wie Aggregations- und Analysefunktionen, um eine Facettisierungen von Attributen und um eine gesteuerte Vererbung.

Das für CAD-Prozesse die Sicht der Objektorientiertheit die gegebene ist, wurde bereits gezeigt. Es sollte daher auch für AutoLisp, als einer häufig genutzten Programmierschnittstelle der CAD-Standardsoftware AutoCAD, ein intuitives, dem CAD-Prozeß angepaßtes LISP-Objektsystem zur Verfügung stehen. Dies schließt die Verfügbarkeit der Aggregationsrelation ein. Es böte die Grundlage für das Experimentieren objektorientierter Modellierungstechniken in AutoCAD. Mit dem System 'OBJECTIVE CAD'[8] wurde dies ebenfalls angestrebt. Es beschränkt sich jedoch auf visualisierbare Objekte, da es die Zeichnungsdatenbank von AutoCAD zur Objektverwaltung nutzt (erweiterte Objektbeschreibungen in 'extended entity data').
Basierend auf den funktionalen Anforderungen an ein dynamisches MVS wurde ALOS entwickelt. Das gesamte System besteht (gegenwärtig) aus den folgenden Modulen:

- **ALOS** - Basismodul, stellt grundlegende Datenstrukturen und Funktionen bereit
- **ALAS** - Aufbau, Update und Analyse von Aggregationsstrukturen
- **ALMS** - System zur Implementation von Objektmonitoren
- **ALRES** - vorwärtsverkettende Regelinterpreter

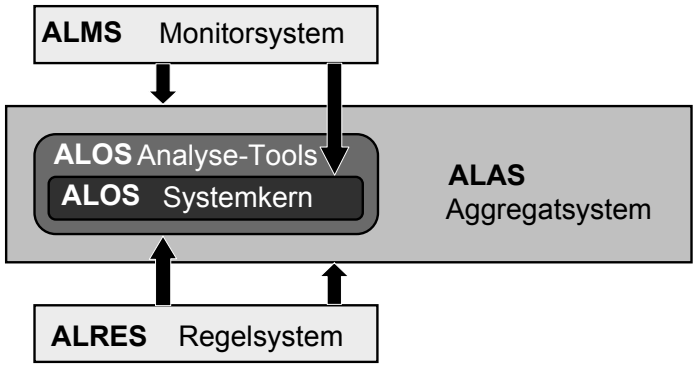


Abb. 8-5 Modul-Architektur des AutoLisp-Objektsystems ALOS

[7] CommonLisp Objekt System
[8] System der Firma IBB, Kassel

8.2.1 ALOS - Basisfunktionen zur Objektverwaltung

Die wesentlichen Merkmale von ALOS (Basismodul) sind:

- Offener, in AutoLisp implementierter Objektaufsatz
- Hierarchisches Klassensystem mit einfacher Vererbung
- Laufzeitdynamische Objektstrukturen mit 'echten' Klassenobjekten
- Funktionen zum Lesen und Schreiben von Objekten sowie Taxonomien
- Analysefunktionen für Taxonomien zur Unterstützung der Programmierung.

Alle Funktionen des Objektsystems werden als Lisp-Funktionen direkt genutzt. Im Sinne des OO-Konzeptes sollten sie eigentlich als Methoden eines Objekts genutzt werden können, um die Polymorphie im Rahmen der Erbschaft zu nutzen. Aus Gründen der begrenzten Ressourcen unter AutoLisp und mangelnder Effizienz wurde jedoch vorerst darauf verzichtet.

In ALOS wird zwischen Objektidentifikatoren (OID) und Objektnamen unterschieden. Die Objektidentität ist über Surrogat-OID's gewährleistet, da diese vom System unique generiert werden (Einschränkung: unique nur im Rahmen einer Session und max. 400000 OID). Diese OID's sind Symbole des LispSystems, sie enthalten in ihrer Value-Zelle alle (lokalen) Informationen des Objektes in Form einer ASSOC[9]-Liste. Methoden in Form von LAMBDA-Ausdrücken werden aus Effizienzgründen nicht in diesen Listen vorgehalten, sondern lediglich deren Methodensignaturen. Diese Signaturen sind wiederum unique Identifikatoren, die als Symbole den eigentlichen Methodenkörper des LAMBDA-Ausdrucks enthalten. Die folgenden Symbole sind systemintern und dürfen per Konvention nicht zur Applikationsprogrammierung genutzt werden:

- `$Inst0, $Met0` Zählervariable zur Generierung der Surrogate
- `$IAAAA...$IXXXX` Symbole für Instanzobjekte / OID's
- `$MAAAA...$MXXXX` Symbole für LAMBDA-Ausdrücke / Methodensignaturen
- `$Itab0` Symbol zur Speicherung der Mappingtabelle Instanznamen auf OID's

Die Wurzel der Taxonomie bildet das Objekt '`ROOT`. Das Objektsystem unterstützt sowohl Struktur- als auch Wertvererbung. Als grundlegender Erbschaftsmechanismus wurde dynamische Erbschaft ('lazy inheritance', [CUN95]) implementiert, d.h. Slots (Werte und Methodensignaturen) werden beim Zugriff durch Suche ermittelt. Dies bietet eine redundanzarme Speicherung, und es ist im Falle des Änderns eines Wertes bei einer Klasse keine Propagation des Wertes zu Subklassen bzw. Instanzen erforderlich. Da dies allerdings in Abhängigkeit der Tiefe der Taxonomie zu starken Laufzeitverlusten beim Zugriff führt, wird zumindest dem Applikationsentwickler die Möglichkeit geboten, sowohl Klassen als auch Instanzen statisch zu erzeugen, d.h. es werden alle verfügbaren Slots für ein neues Objekt bei dessen Erzeugung kopiert.

Der ALOS-Basismodul realisiert die orthogonalen Kernfunktionen. Dieser Kern wurde erweitert um Funktionen zur Analyse der Taxonomie und der Objektstruktur, da sie in Applikationen häufig benötigt werden. Folgende Gruppen von Funktionen stehen zur Verfügung:

- *Erzeugen, Löschen und Umhängen von Objekten* (Klassen und Instanzen)
- *Erzeugen, Löschen und Ändern von Slots*
- *Zugriff auf Objekte und zum Nachrichtenaustausch*
- *Lesen und Schreiben von Objekten*

[9] Liste von Schlüssel-Wert Paaren

- *Prädikative Funktionen zum Prüfen von S-Ausdrücken*
- *Fehlerfunktion*
- *Strukturanalyse der Objekthierarchie*
- *Strukturanalyse von Objekten*
- *Anzeigefunktionen* (Darstellen von Objekten bzw. des Objektbaumes, alphanumerisch)

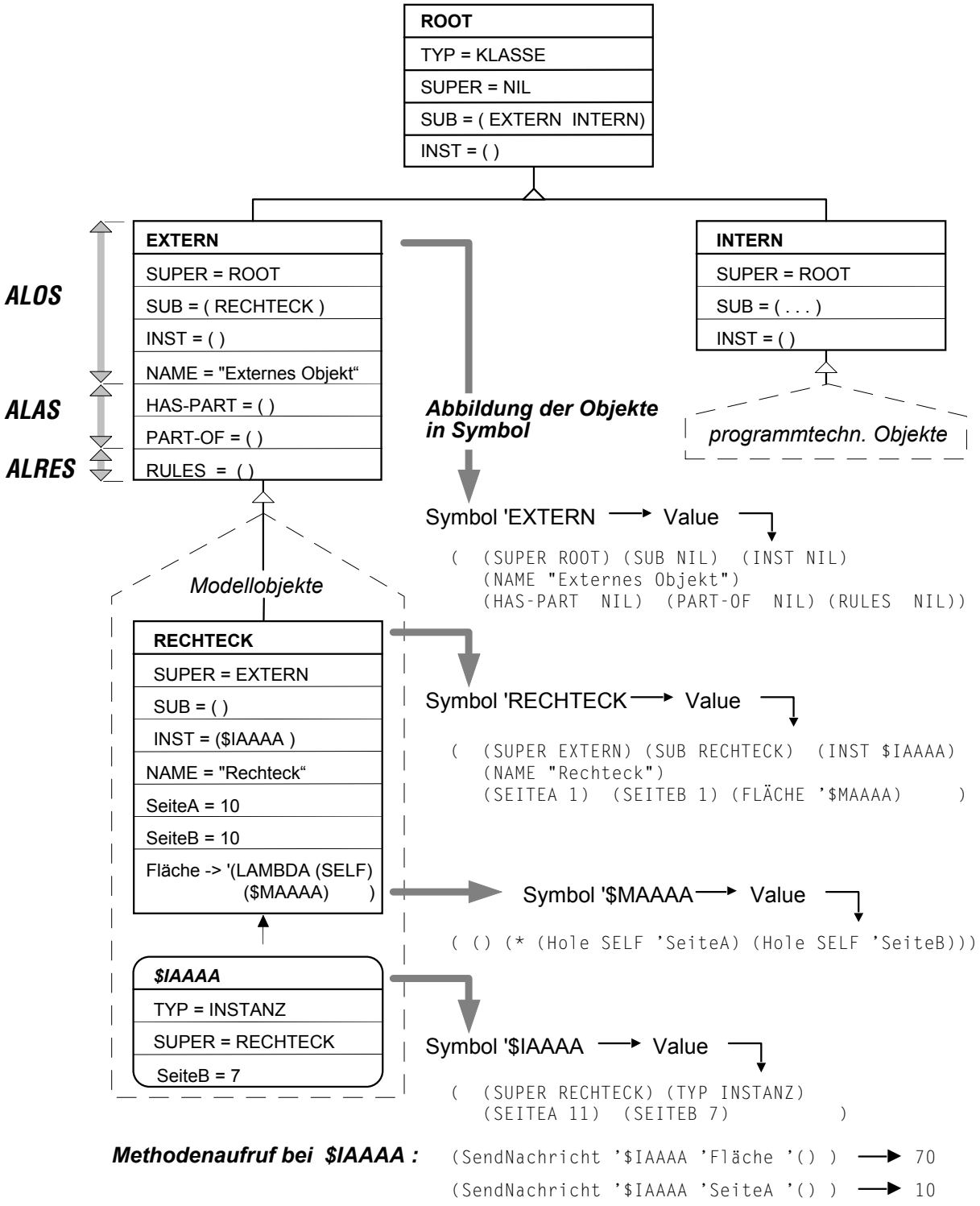


Abb. 8-6 LISP-Datenstruktur des ALOS-Kerns

8.2.2 ALAS - AutoLisp Aggregatsystem

Der ALAS-Modul macht Funtionalität zur Erzeugung, Verwaltung und zur Analyse von Aufbaustrukturen verfügbar. Die Mnemonik der Funktionsrufe unterscheidet zwei verschiedene Objekttypen, die durch ihre Stellung im Aggregationsgraphen erkennbar sind: *Simplexe* und *Komplexe* (siehe Glossar, Anhang A). Die Namensgebung lehnt sich an die bei LAABS ([LAA91]) an, die dort allerdings spezieller gefaßt ist.
Durch Laden des ALAS-Moduls verfügen alle Objekte ab der Klasse 'EXTERN über den Slot (Relationen) 'PART_OF (Liste der Komplexobjekte) sowie über 'HAS_PART (Liste der Teilobjekte), Aufbaustrukturen können also heterarchisch sein.
Es sei hier noch einmal darauf verwiesen, daß die Semantik von Relationen (aller Relationen) bei Klassen eine andere als bei Instanzen ist. Ein konkretes Komplexobjekt (Instanz) ist aus konkreten Instanzen aufgebaut! Im Gegensatz dazu verweist der 'HAS-PART Slot bei einer Komplexklasse auf andere Klassen, deren Instanzen potentielle Teile von Instanzen der Komplexklasse sein könnten. Wertvererbung zwischen Instanzen und Klassen macht hierbei keinen Sinn! Folgende Arten von Funktionen stehen zur Verfügung:

- *Funktionen zum Einbau, Ausbau und Umbau von Objekten*
  (d.h. auch ganzer Teilbäume)
- *Funktionen zur Analyse der Aufbaustruktur*
- *Prädikative Funktionen*
- *Lese- und Schreiboperationen*
- *Anzeigefunktionen* (alphanumerisch)

8.2.3 Propagation im AutoLisp-Monitorsysteme ALMS und dem AutoLisp-Regelinterpreter ALRES

Klassenmonitore sind ein effizienter Weg zur Triggerung von Folgeaktionen beim Erzeugen und Löschen von Instanzobjekten (im ALOS-Basismodul) sowie bei ihrem Einbau und Ausbau in der Aggregationsstruktur (in ALAS). Auf 'echte' Slotmonitore wie sie KappaPC kennt, wurde verzichtet, da eine weitere Verlangsamung des Slotzugriffs die Folge wäre. Da ALOS Slots in ASSOC-Listen realisiert werden, kennt es keine Slotobjekte und damit keine Facettisierung von Slots. Slotmonitore wären also über eine Konvention für die Expansion der Slotnamen zu realisieren gewesen (siehe Kap. 7). Dies war aus Effizienz- und softwaretechnischen Gründen inakzeptabel. Die Monitore werden über spezifische Methoden der Klassen implementiert. Der ALMS-Modul bietet folgende Monitore:

- `BeiErzeugen` - Klassenmethode, feuert nachdem die Instanz erzeugt wurde
- `BeiLoeschen` - Klassenmethode, feuert bevor die Instanz gelöscht wird
- `BeiEinbau` - Instanzmethode, feuert nachdem die Instanz eingebaut wurde
- `BeiAusbau` - Instanzmethode, feuert bevor die Instanz ausgebaut wird

Zum Erzeugen der Monitore dient eine einzige ALMS -Funktion:

`(ErzeugeKlassenMonitor` *`classname monitor methodenname`*`)`

Die Klasse *`classname`* wird mit einem Monitor *`monitor`* versehen. Die Methode *`methodenname`* wird entsprechend des *`monitors`* getriggert. Die Methode muß parameterfrei sein. Beim Laden des ALMS-Moduls werden die Funktionen '`ErzeugenInstanz` und '`LoescheInstanz` sowie '`BaueEin` und '`BaueAus` entsprechend überladen.

Der Modul **ALRES** implementiert einen vorwärtsverketten Regelinterpreter mit steuerbarer Suchstrategie. Er bietet dem Anwendungsprogrammierer die Möglichkeit, mit regelbasierten Problemlösungsmechanismen zu experimentieren. Im Gegensatz zu KAPPAPC oder NEXPERTOBJECT werden Produktionsregeln jedoch um einen 'ELSE'-Zweig erweitert. Regelausdrücke werden ähnlich zu Methoden realisiert, wobei für Prämisse und die beiden Konklusionsteile ('THEN'-Teil, 'ELSE'-Teil) getrennt spezielle LAMBDA-Ausdrücke konstruiert werden. Die Ablage einer Regel erfolgt in deren Namenssymbol, später wären jedoch auch speziell konstruierte Identifikatoren (Regelsignatur) sinnvoll. Im Gegensatz zu KAPPAPC erfolgt über die obligatorischen 'RulePattern eine enge Bindung an Objektklassen (Eintragen in Slot 'RULES aller betroffener Objekte), so daß effizient Mengen relevanter Regeln gebildet werden können (siehe KAPPAPC-Manual [KAP20]). In einem Bindevorgang wird ein Iterationsausdruck gebildet, in dem nur die Prämissen von Regeln ausgewertet werden, die im 'RULES'-Slot verzeichnet, d.h. relevant sind. Für Regeln, deren Prämisse zu True evaluiert, wird der THEN-Teil gebunden und evaluiert, bei False der ELSE-Teil (siehe Beispiel in Abb. 8-7). Grundmodell des Verkettungsmechanismus ist eine datengetriebene Propagation, d.h. bei Veränderung eines Objektes werden die in seinem 'RULES'-Slot verzeichneten Regeln der Menge (Liste in $ARULE) der relevanten Regeln hinzugefügt. Der Prozeß terminiert, wenn die Liste leer ist oder eine GOAL (prädikativer Ausdruck) erfüllt ist (Algoritmus siehe Abb. 8-8).

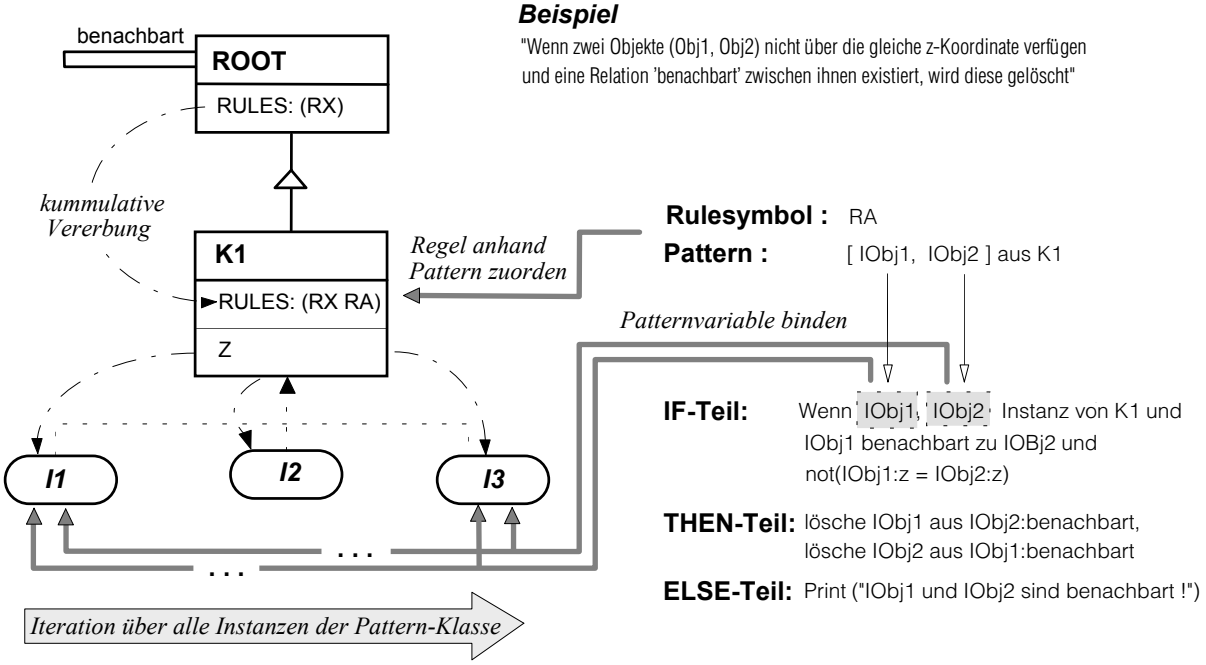


Abb. 8-7 Bindungsmechanismus vorwärtsverkettender Regeln in ALRES

Zur Nutzung des Regelsystemes wird ein globales Symbol $ARULE (Rulestack) erzeugt, das die Liste noch auszuwertender Regeln ('relevant rules') enthält. Es darf im weiteren nicht zur Programmierung verwandt werden.

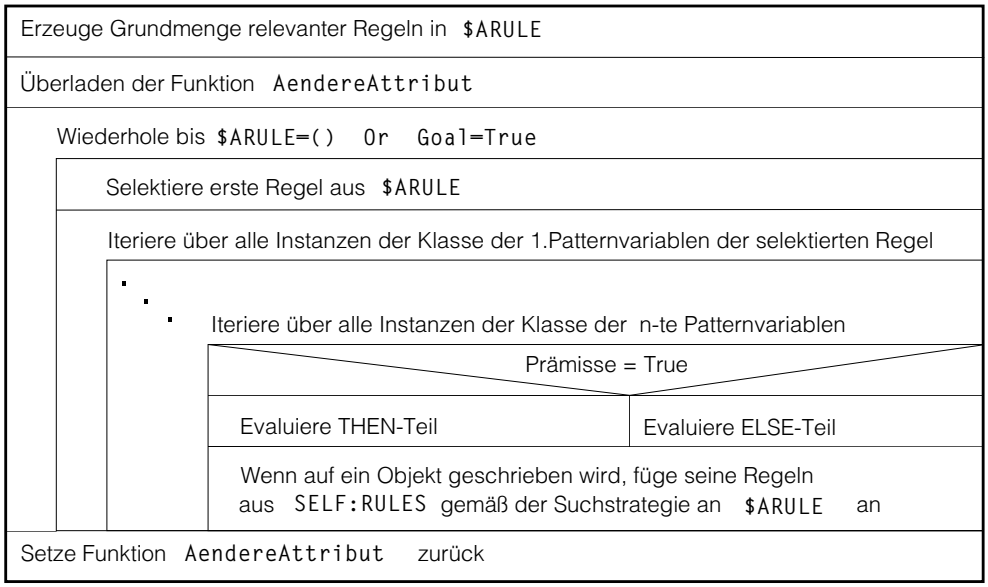


Abb. 8-8 Algorithmus der Regelpropagation in ALRES

Nutzbare Funktionen der ALRES Moduls sind:

- *Erzeugen von Regelausdrücken und deren Eintrag in die entsprechenden Slots*
- *Starten des Propagationsprozesses unter Angabe eines Goals*
- *Steuerung der Suchstrategie*
- *Protokollfunktionen* (alphanumerisch).

Die Implementation ungerichteter Constraints wurde untersucht. Sie ist in eingeschränktem Maße möglich, wurde jedoch auf Grund mangelnder Effizienz verworfen.

8.2.2 Bewertung ALOS

ALOS hat sich als Experimentiertool bewährt und liefert als AutoCAD-Erweiterung eine Effizienz, die der einer kommerziell verfügbaren OO-Entwicklungsumgebung für PC vergleichbar ist (siehe Test Anhang F). Lediglich bei der Destruktion von Objekten schlägt der offensichtlich ineffektive Garbage-Collector von AutoLisp durch. Die Module laufen stabil und wurden bereits zu weiteren Applikationsentwicklungen, wie für objektorientierter Konstruktionssoftware im Stahlbau ([MAS95]) verwandt.

Testbeispiel:

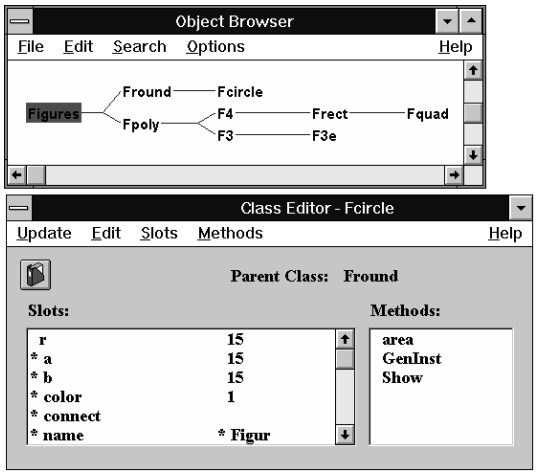
A - Aus einer Modellwelt von 2½D-Prismen, die durch 9 dynamisch attributierte Klassenobjekte beschrieben wird, werden 5000 Instanzen nach Zufallsprinzip erzeugt, mit je ca. 6 Slots dynamisch attributiert und ca. 50 % der Attribute lokal initialisiert. Eine Relation erzeugt eine sequentielle Liste aller Instanzen.

B - Alle Instanzen und Relationen werden entlang der Relation gezeichnet.

C - Entlang der Relation werden in Abhängigkeit eines numerischen Attributes ca. 50% der Instanzen selektiert, deren Masse und die Gesamtmasse berechnet.

D - Alle Instanzen werden entfernt.

Testumgebung: Pentium100, 24 MBRam, Windows 3.11
(vollständige Testergebnisse im Anhang F)




```
Befehl: (ZeigeHierarchie 'EXTERN)
  EXTERN :
    FIGUR :
      FROUND :
        FCIRC :
      FPOLY :
        F3 :
          F3E :
        F4 :
          FRECT :
            FQUAD :
```

```
Befehl: !FCIRC
( (TYP KLASSE) (NAME "KREIS")
  (SUPER FROUND) (SUB NIL) (INST NIL)
  (H 2) (RO 2.6) (COLOR 1) (R 7)
  (AREA (LAMBDA (SELF / R) ($MAAAJ)))
  (SHOW (LAMBDA (SELF) ($MAAAI)))
  (GENINST (LAMBDA (SELF / TINST) ($MAAAH)))
)
```

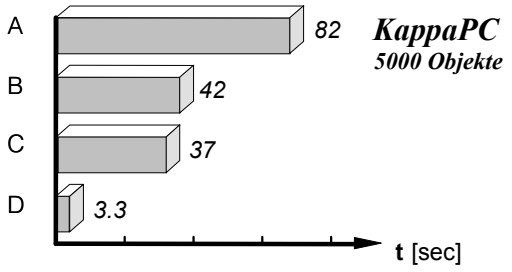


Abb. 8-9 a: Laufzeittest für eine dyn. Modellwelt aus 2½D-Flächen in KappaPC

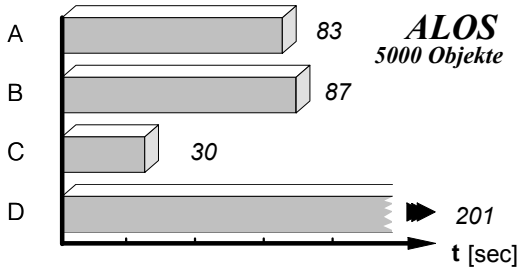


Abb. 8-9 b: Laufzeittest für eine dynamische Modellwelt aus 2½D-Flächen in ALOS

8.3 FLEXOB – C++ basiertes flexibles Objektsystem

Das vorgestellte AutoLISP-Objektsystem hat bedingt durch seine interpretative Arbeitsweise eine Laufzeiteffizienz, die es für große Projekte ungeeignet erscheinen läßt. Bei der Weitergabe von ALOS-Programmen ist die existentielle Abhängigkeit von AutoCAD sowohl technisch als auch monetär problematisch. Als Konsequenz führt dies zur Entwicklung eines kleinen, im Funktionsumfang auf das Anforderungsspektrum passend zugeschnittenen, flexiblen Objektverwaltungsmoduls.

Als Wirtssprache wurde aus mehreren Gründen C++ gewählt:

- OBJEKTORIENTIERTHEIT: C++ verfügt über die wichtigsten Eigenschaften OO-Programmiersprachen (Klassen, Vererbung, Polymorphie, late binding, generischer Code, ...)
- PORTABILITÄT: C++ Compiler stehen für praktisch jedes relevante Betriebssystem (und Hardwareplattform) zur Verfügung
- POPULARITÄT: C++ ist eine Programmiersprache mit hoher Nutzerakzeptanz und breiter Unterstützung (Development-Tools, Klassenbibliotheken und OO-Datenbanken usw.).

Die Wahl einer compilierenden Wirtssprache und dem damit einhergehenden Verzicht auf die Möglichkeiten der Metaprogrammierung (wie bei ALOS) bedingt eine *semidynamische* Arbeits- bzw. Nutzungsweise. Aufgrund der semidynamischen Arbeitsweise des Modellierkerns zerfällt seine Nutzung in zwei Phasen (vergl. WEHNER [WEH95]):

I. *Statische Phase (Compilezeitphase)*
Diese Phase dient der domänenspezifischen Konditionierung des Modellierers. Alle Attribut- und Relationenklassen, die für die Produktmodellerstellung in einer bestimmten Anwendungsdomäne benötigt werden, sind als Code in C++-Klassen zu implementieren. Ebenso muß der Applikationsprogrammierer alle die (wurzelnahen) Klassen der Taxonomie implementieren, die:
- Methoden enthalten, die für die Domäne spezifisch sind (prozedurales Wissen)
- Methoden für Berechnungen über Objektmengen enthalten
- Methoden für Iterationen über Aggregationsstrukturen enthalten.

Das Basisobjektverwaltungsmodul stellt dazu (Meta-) Klassen bereit, die bereits die gesamte Funktionalität zur Objektverwaltung enthalten. Der Applikationsprogrammierer muß seine domänenspezifischen Modellklassen von diesen Klassen ableiten. Es entsteht ein C++-Klassensystem, das für die Nutzung in der nächsten Phase dynamisiert wird.

II. *Dynamische Phase (Laufzeitphase)*

In dieser Phase erfolgt die Erweiterung des Domänenmodells durch Eingabe des zugehörigen deskriptiven Wissens. Hierzu wird das Modell durch Spezialisierung bereits in der Taxonomie definierter Klassen erweitert. Subklassen mit gleichem Verhalten einer implementierten Superklasse können rekursiv dynamisch spezialisiert werden. Diese Subklassen können durch Hinzufügen von Attributen und Relationen (Slots) sowie durch Ändern von Wertebelegungen bzw. Einschränken von Wertemengen geändert bzw. erweitert werden. Slotobjekte werden durch Instanziierung ihrer Slotklassen gebildet, mit Werten belegt und zugeordnet. Die Projektdaten werden durch Instanziierung von Modellklassen, ggf. Änderung der ererbten Standardwerte und vor allem durch den Aufbau einer (heterarchischen) Aggregationstruktur erzeugt.

8.3.1 Basisdatenstrukturen / Systemarchitektur

Der FLEXOB-Kern besteht aus einem Satz orthogonaler Elementarfunktionen. Komplexe Funktionen lassen sich auf Folgen dieser Elementarfunktionen abbilden. Diese Kernfunktionalität unterteilt sich in die Kategorien:

- *Konstruktion von Objekten*
- *Destruktion von Objekten*
- *Zugriff auf Objekte* (Klassen, Instanzen, Attribute, Relationen und Facetten)
- *Informationsfunktionen* (zur Objekt-, Taxonomie- und Aggregationsstruktur)
- *Objektpersistenz*

Aus Portabilitätsgründen wurde auf die Nutzung kommerzieller Klassenbibliotheken (tools.h++[11], MFC[10], ...) verzichtet.

[10] Applikationsneutrale Klassen verschiedener Hersteller, die Funktionalität zur Listenverwaltung, I/O und vielem mehr kapseln

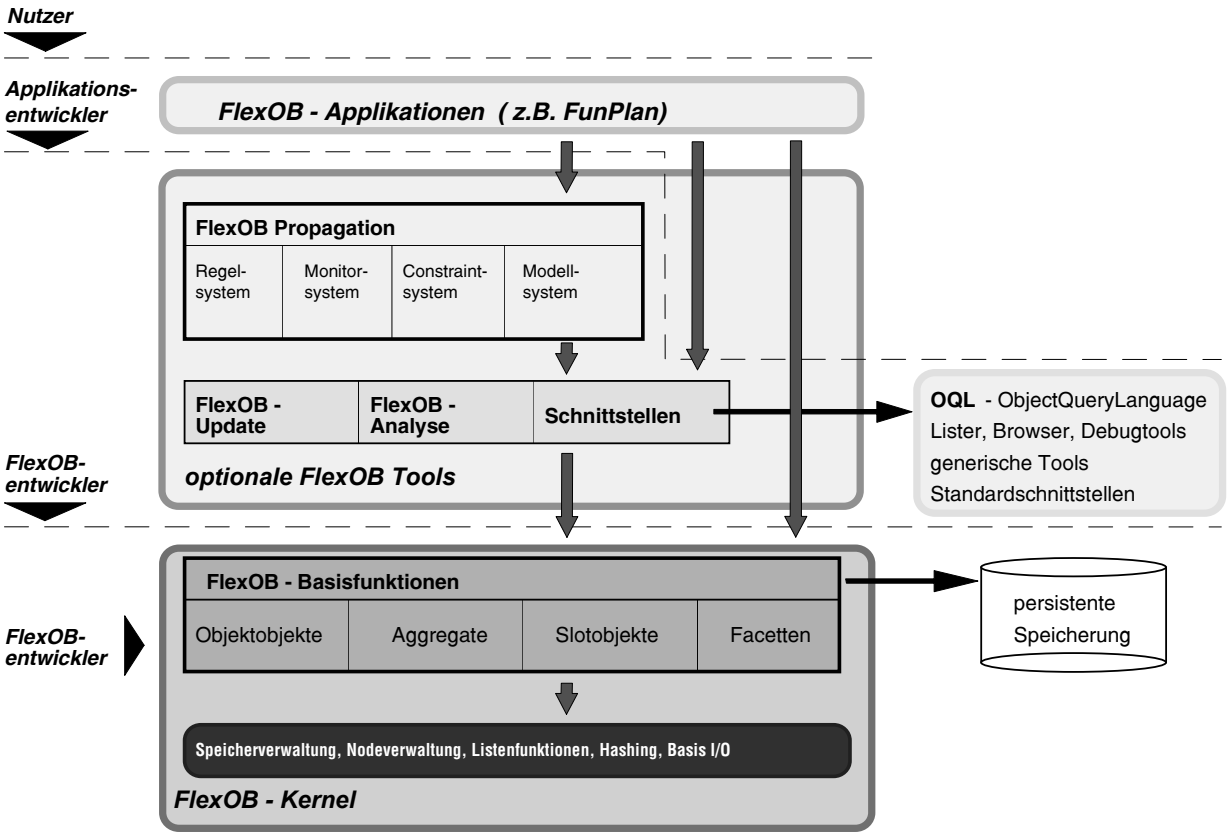


Abb. 8-10 FLEXOB-Systemkonzept

Die Implementierung des FLEXOB-Kernels geht von folgenden Prämissen aus:

- die Wissensorganisation erfolgt in Taxonomien mit einfacher Vererbung
- nur beschreibendes, kein prozedurales Wissen dynamisch erzeugbar
- möglichst hohe Laufzeiteffizienz auch bei großer Anzahl zu verwaltender Objekte
- alle Objekte werden im Hauptspeicher verwaltet (kein Sekundärspeichermanagement)
- Objekte haben starke Objektidentität
- hohe Zugriffseffizienz unter der Annahme $N_{Destruktion} < N_{Konstruktion} << N_{Zugriff}$

Um die Entwicklung von Applikationen zu unterstützen, wäre es hilfreich, wenn Funktionen, die auf dem Metamodell standardisierbar sind, auch standardisiert verfügbar wären. Dies betrifft vor allem eine erweiterte Modell- bzw. Projektanalyse sowie Funktionen zum erweiterten Modell-Update (Reklassifikation, Kopieren ganzer Teilbäume von der Aggregation bzw. der Taxonomie usw.) Diese sind gegenwärtig nicht vollständig implementiert. Ebenso kann die Funktionalität standardisierter Schnittstellen zum Datenaustausch auf dem Kernsystem implementiert werden. Für die STEP-Norm wird daran gerade gearbeitet.

8.3.2 Dynamische Modellobjekte – Extensionen statischer Metaobjekte

Die Metaebene besteht ausschließlich aus statischen C++-Klassen. Sie stellen die allgemeinen Dienste bereit, die zur Laufzeitobjektverwaltung und zur Generierung von Analyseinformationen benötigt werden. In dieser Ebene muß sämtliches ablaufsteuerndes (prozedurales) Domänenwissen eingebracht werden, da zur Laufzeit diese Möglichkeit nicht mehr besteht. Obwohl die Elemente der Wissens- und der Projektebene logisch durchaus eine unterschiedliche Semantik haben, sind sie physisch alle Instanzen einer C++-Klasse

oder genauer, alle dynamischen Klassen- und Instanzobjekte eines Teilbaumes der Objekthierarchie sind Instanzen ein und derselben C++-Klasse (Abb. 8-11). Die Metaebene stellt somit physische Datenobjekte zur Beschreibung von logischen Wissensobjekten bereit.

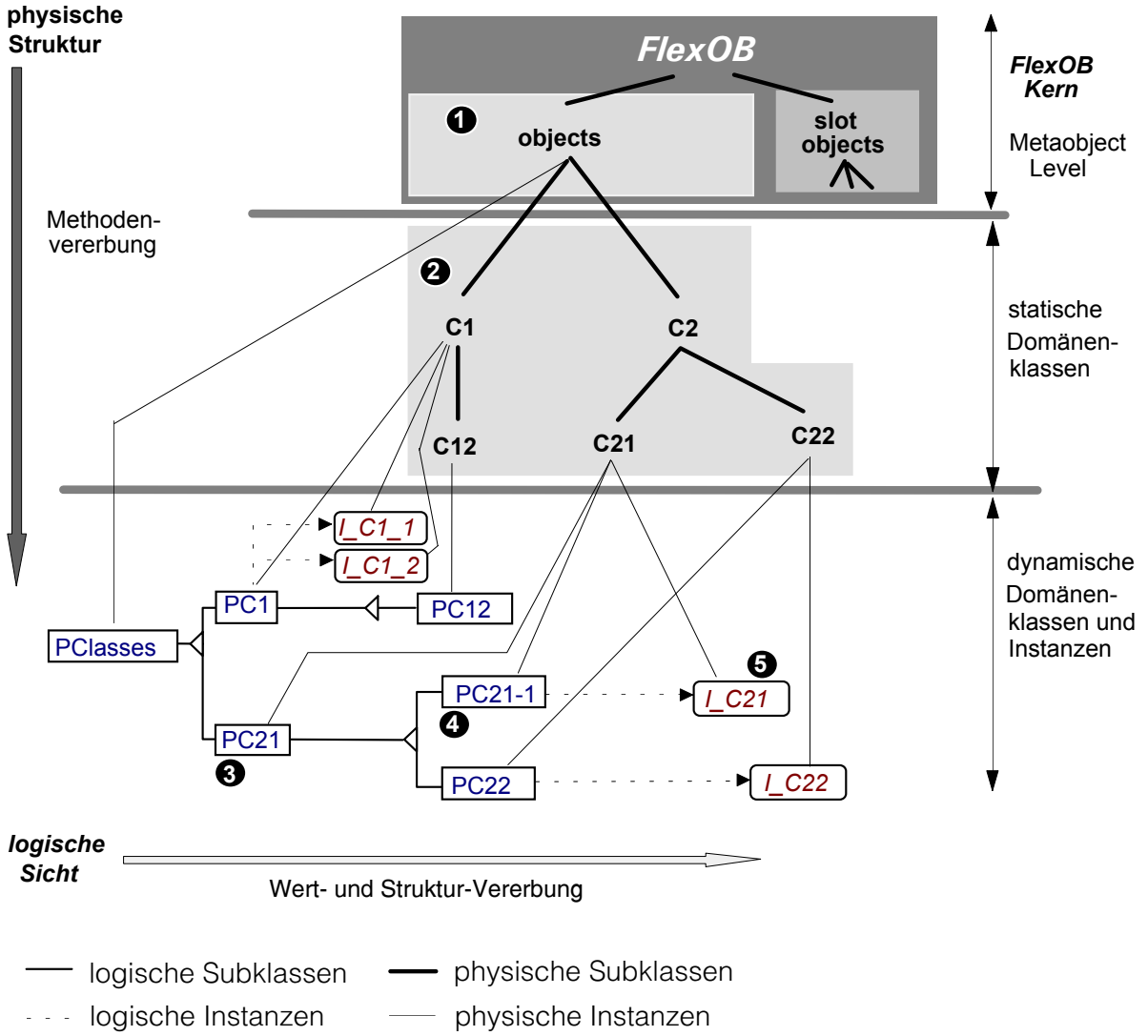


Abb. 8-11 Darstellung dynamischer Objekte

❶ statische C++ Klasse, definiert die allgemeine Struktur von Datenobjekten

❷ statische C++ Klasse, definiert domänenspezifische Daten und Methoden (prozedurales Wissen) sowie Daten und Methoden, die nicht zur Wissensmodellierung gehören (z.B. zur Visualisierung)

❸ dynamische Klasse (logische Klasse, Klassenprototyp für ❷), Eigenschaftsbeschreibung mittels laufzeitgenerierter Attribute und Relationen (deklaratives Wissen)

❹ dynamische Klasse (logische Subklasse von ❸, physische Instanz von ❷)

❺ dynamische Instanz (logische Instanz von ❹, physische Instanz von ❷), konkreter Entitäten der Modellwelt

Eine besondere Bedeutung kommt dabei dem Ebenenübergang von der Metaebene zur Wissensebene zu. Durch diese prototypische Instanziierung wird das ursprünglich statische C++-Objektmodell semidynamisiert. Diese Dynamisierung ist deshalb von partieller Natur, da sie auf die Laufzeitgenerierung zusätzlicher Attribut- und Relationenobjekte beschränkt ist.

Bedingt durch die Verfügbarkeit von 'echten' Klassen- und Slotobjekten geht der FLEXOB-Modellierkern in seiner Fähigkeit zu Objekt-, Taxonomie- und Aggregationsanalyse deutlich über das hinaus, was man mit Laufzeit-Typidentifikation[11] nach ANSI X3J16 oder [STR91] erreichen kann. Klassen in C++ haben einen rein deklarativen Charakter, sie treten im laufenden Programm nicht als Objekt in Erscheinung. RTTI kann sich also immer nur auf Instanzobjekte beziehen.
Für FLEXOB wird die Bereitstellung dieser Informationen dadurch erreicht, daß die C++ Klassen der Domänenklassenhierarchie zu 'echten' Objekten gemacht werden, indem sie in eine Hierarchie von physischen Prototypen überführt werden (siehe Abb. 8-11). Alle Aspekte der Methodenbereitstellung und Methodenerbschaft werden von C++ unterstützt. Die zusätzlichen Dienste wie Werteerbschaft und Objektanalyse stellt der FLEXOB-Kern bereit.

8.3.3 Die Klasse FOObject - Implementation von Metaobjekten

Klassen und Instanzen der Domäne verfügen als Objekte über die gleiche Struktur. Sie werden deshalb beide von der Klasse FOObject abgeleitet[12]. In ihr werden dynamische Eigenschaften bereitgestellt, die deutlich über die des C++-Laufzeitsystems hinausgehen.
Der Nutzer kann zur Laufzeit Struktur- und Metainformationen zu Objekten ermitteln, wie:

- Super- und Subklassen einer Klasse
- Instanzen einer Klasse sowie die Klasse einer Instanz
- Attribute und Relationen (Slots) einer Klasse oder Instanz
- Analyse der Aggregationsstruktur einer Instanz bzw. seines Komplexes.

Die explizite Verwaltung der Slots in einer Slottabelle ermöglicht es dem Nutzer, zur Laufzeit Attribute und Relationen zu seinen Klassen- und Instanzobjekten hinzuzufügen, die in den statischen Domänenklassen noch nicht bekannt waren (dynamische Spezialisierung). Weitere Aufgaben von FOObject sind die:

- Bereitstellung eines geeigneten Mechanismus für die strukturelle Erbschaft
- Identifikation von Slots und Herstellen der Laufzeitbindung
- konsistente Namensvergabe und -verwaltung
- Ein- und Ausgabe (Objektpersistenz)
- Erzeugung des initialen Objektbaumes (Prototyperzeugung).

Einen umfassenden Überblick über den der Implementation zugrunde liegenden Klassenbaum zeigt die Darstellung in Abbildung 8-12.

[11] Runtime Type Information, bei Borland RTTI
[12] Eine Aufteilung in zwei Klassen (z.B. FOClass und FOInstance) hätte die Doppelung des Domänenmodells zur Folge.

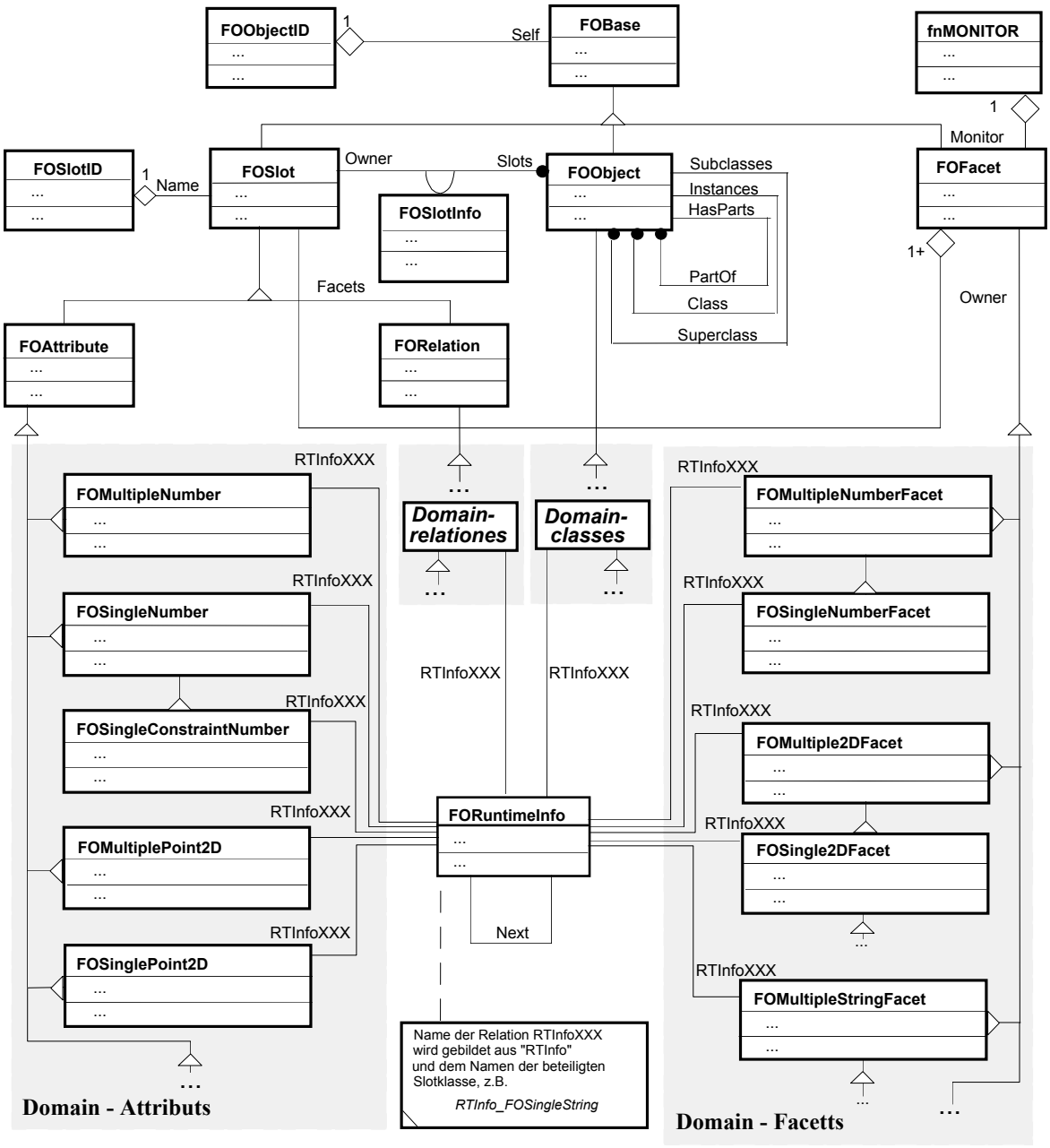


Abb. 8-12 Klassendiagramm des FLEXOB-Modellierkerns (nach [STE97])

8.3.4 Slotobjekte - Implementierung der Wertvererbung

Alle paradigmatischen Relationen ([SAG91]) werden aus Effizienzgründen explizit vom Objektverwaltungsmodul unterstützt, d.h. sie werden gemäß der Anforderung durch Membervariablen der 'ObjectObject'-Klasse realisiert. Diese Membervariablen referenzieren nur auf MetaObjekte. Alle anderen Relationen, deren Semantik der Nutzerinterpretation unterliegt (Assoziationen), werden durch Relationenobjekte abgebildet.

Es ergibt sich somit für Slotobjekte ebenfalls ein Typbaum (Abb. 8-13), der im Gegensatz zu den Klassen nicht *dynamisch* spezialisiert werden kann. Es muß die gesamte, für eine bestimmte Anwendungsdomäne geforderte Varietät von Slottypen mittels statischer C++-Klassen implementiert werden. Zur Laufzeit können von diesen Klassen physische Instanzen

erzeugt werden. Diese physischen Attributobjekte werden dann Anwendungsklassen bzw. deren Instanzen zugeordnet und ggf. ihrer Wertebelegung geändert ([WEH95]).

In einigen Objektsystemen (wie auch in ALOS) werden Attribute als Paare (Name, Wert) in Assoziativlisten gespeichert. Im Gegensatz dazu werden im FLEXOB-Kernel Attribute und Relationen als eigenständige Objekte implementiert. Daraus ergeben sich folgende Vorteile:

- *Softwaretechnisch:* Das Konzept der Objektorientiertheit wird konsequent auch auf Attribute und Relationen übertragen. Die gesamte Slotsemantik inklusive der Zugriffsmethoden ist in diesen Slotobjekten und nicht im entsprechenden Klassen- oder Instanzobjekt enthalten.
- *Laufzeittechnisch:* Es kann ein Erbschaftsalgorithmus implementiert werden, der die Vorteile des schnellen Zugriffs der statischen Erbschaft (eager inheritance, [CUN95]) und der rationellen Datenhaltung der dynamischen Erbschaft (lazy inheritance) vereinigt.

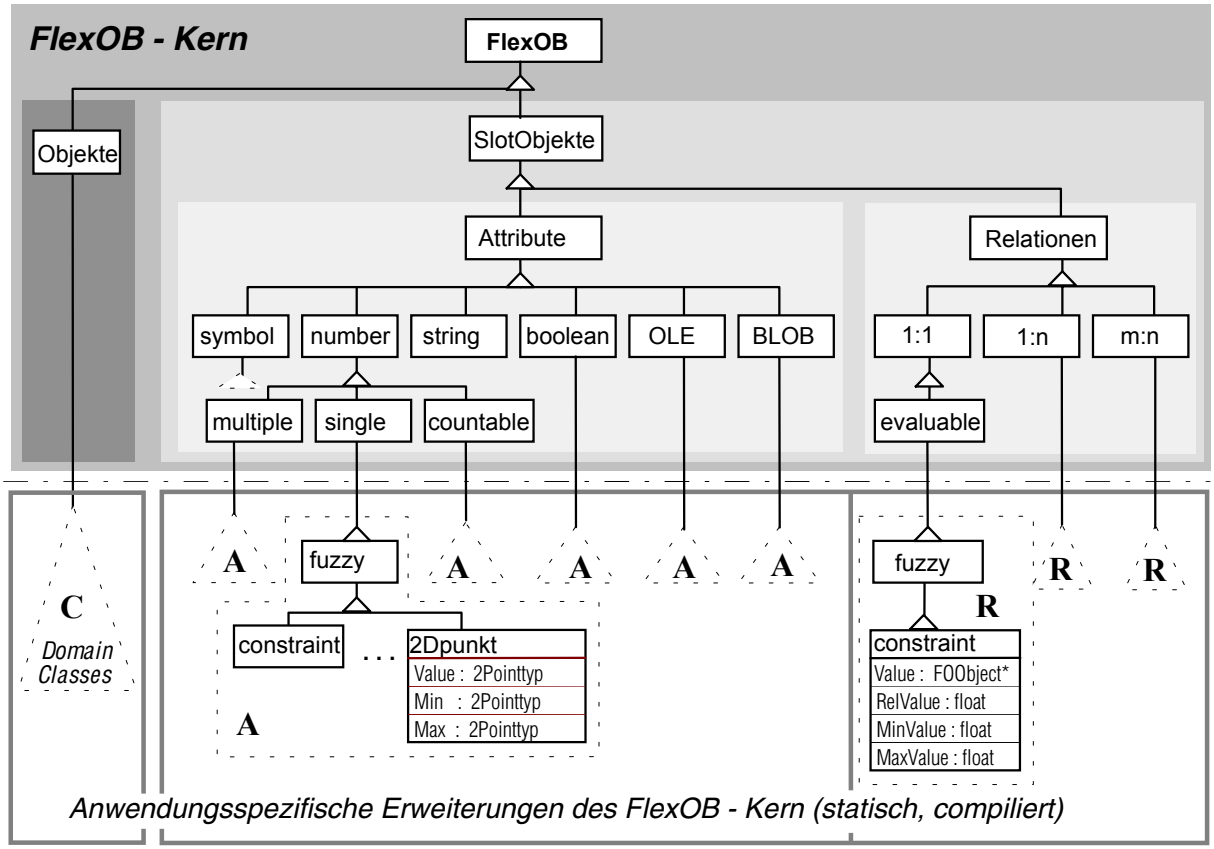


Abb. 8-13 Mögliche Ausprägung des statischen Klassensystems für Slotobjekte

Jedes Klassen- bzw. Instanzobjekt verfügt über eine lokale Slottabelle (Slotidentifikator, Slotreferenz). Beim Erzeugen einer neuen Klasse oder Instanz wird diese Tabelle von der Superklasse bzw. Klasse kopiert. Damit erbt das neue Objekt alle Eigenschaften seines Superobjektes. Es werden also nicht die Slotobjekte selbst kopiert (wie bei statischer Erbschaft), sondern nur Referenzen auf sie.

Folgende Fälle sind beim Slotzugriff zu unterscheiden:

- *Lesender Zugriff*: Der Slotidentifikator des im Scope des zugreifenden Objektes liegenden Slotobjektes wird ermittelt (Laufzeitbindung).
- *Schreibender Zugriff*: Wenn das Slotobjekt lokal ist (Ermittlung über Flag), wird direkt auf das Slotobjekt geschrieben (kein Propagationsaufwand), sonst wird das ererbte Slotobjekt kopiert und die Wertänderung auf der Kopie ausgeführt.
- *Neuanlegen eines Slots*: Für alle Subklassen- bzw. Instanzen des strukturell zu erweiternden Domänenobjekts muß rekursiv ein neues Paar (Slotidentifikator, Slotreferenz) in dessen Slottabelle eingetragen werden. Es werden jedoch nur Referenzen, nicht aber ganze Slotobjekte kopiert.

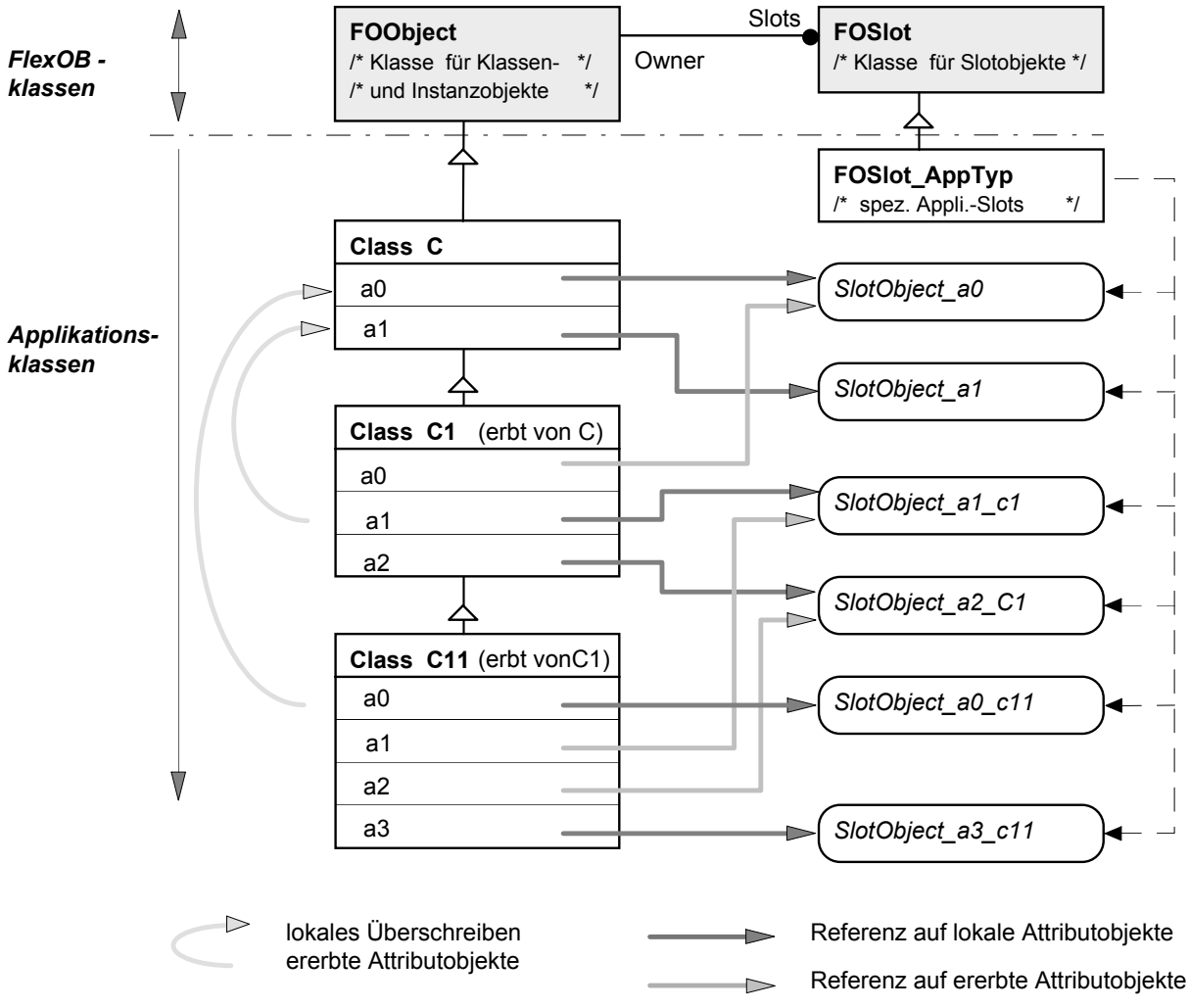


Abb. 8-14 Schematische Darstellung des Erbschaftsmechanismus

In der Klasse FOSlot (Abb. 8-14, vgl. Abb. 8-12) wird eine Slotverwaltung implementiert, die die Konsistenz der Namensverwendung sichert, eine einfache Umbenennung von Slotobjekten ermöglicht, den Speicherplatz für Namenstring nur einmal zuweist sowie eine Vereinfachung der persistenten Speicherung bietet.

8.3.5 Die Klassen FOSlot und FOFacet – Implementierung von Slotobjekten

Jedes Slotobjekt besteht aus einer Aggregation von Facetten, wobei jedoch mindestens ein '`ValueFacet`' existieren muß. Ziel ist es, eine konventionelle Sicht auf Slots zu gewährleisten, die ja keine Facetten kennt. In FLEXOB ist ein Wertezugriff normalerweise zweistufig, als Attribut bzw. Relation fungiert ein Slotobjekt, aus dem der Wert einer Facette extrahiert wird. Um ein konventionelles Interface zu Attribut- und Relationen anbieten zu können, wurde die '`ObjektObjekt`'-Klasse um eine Methode '`GetValue`' erweitert, die den zweistufigen Zugriff auf das '`ValueFacet`' implizit ausführt.

Die Klasse FOSlot stellt folgende Basisfunktionalität bereit:

- FACETTENVERWALTUNG: Implementierung einer allgemeinen Zugriffsfunktion, die via Erbschaft bei abgeleiteten Klassen wiederverwendet wird
- FACETTENZUGRIFF: Ein einheitliches Interface zum Eintragen von Werten in und zum Holen von Werten aus Facetten mittels der Methoden `GetFacet()` und `SetFacet()`. Durch den Eintrag zu erzeugender Facetten auf Facettentabellenpositionen, die der zu erwartenden Zugriffsfrequenz entspricht, werden Suchzeiten beim Zugriff minimiert.
- METAINFORMATIONEN: Wie auch schon für Klassen- und Instanzobjekte, können zu Slotobjekten während der Programmlaufzeit Metainformationen ermittelt werden.
- WERTEERBSCHAFT: Vererbung konkreter Wertbelegungen von Eigenschaften im Gegensatz zur Vererbung der Eigenschaft an sich.
- EIN - UND AUSGABE: Herstellung von Persistenz für Slotobjekte.

Der überwiegende Teil dieser Funktionen kann in allen abgeleiteten Domänenslotklassen ohne Änderung verwendet werden. Damit ist eine leichte Erweiterbarkeit des Attribut- und Relationensystems durch den FLEXOB-Applikationsprogrammierer möglich.

Facetten stellen im FLEXOB-Slotklassenkonzept die Konstrukte dar, die Daten im gebräuchlichen Sinn enthalten. Sie repräsentieren Werte, die durch *Typen* der Wirtssprache beschrieben werden. Zur Schaffung eines einheitlichen Interface, insbesondere zu anderen Programmiersprachen, hat es sich als sinnvoll erwiesen, diese Basisdatentypen auf eine Grundmenge zu beschränken. Für die Domäne „Architektonischer Vorentwurf" (FUNPLAN) wurden folgende Facettentypen als '*single values*' sowie als Liste der Datentypen ('*multiple value*') implementiert:

- `NUMBER (FIXNUM, FLONUM)`
- `STRING`
- `POINT2D, POINT3D`
- `RELATION`

8.3.6 Speicherkonzept und Objektidentität

In der gegenwärtigen Version geht FLEXOB von der Annahme aus, daß die Modelle vollständig im Hauptspeicher verwaltet werden können. Bei weiter steigender Modellgröße wird das Sekundärspeichermanagement des Betriebssystems genutzt. Die an sich effiziente Speicherung in einer OO-Datenbank wurde unter den getroffenen Annahmen verworfen, da dies einen zu hohen Implementationsaufwand bedingt hätte. Weite Teile des Leistungsspektrums solcher Datenbanken, wie Sekundärspeichermanagement, Transaktionsmechanismen etc. werden für die beabsichtigte Neuimplementierung von FUNPLAN gegenwärtig nicht benötigt.

Für andere CAD-Systementwicklungen sind sie allerdings durchaus wünschenswert, insbesondere für die Verteilung von Modellen und Projekten nach dem Client-Server Prinzip sowie für die Verwaltung großer Projektmodellbestände. Die erforderlichen Dienste sind als Schicht auf dem Kernsystem implementierbar. Auch das Design von OO-Datenbanken würde in diesem Fall vom Gedanken der Metaobjekte profitieren. Die Datenbankstruktur muß bei entsprechenden Domänenmodellupdates nicht geändert werden, da sie nur semantikfreie generalisierte Objekte verwahrt. Lediglich das Hinzufügen von Slotklassen (statischen Nutzungsphase) würde eine Erweiterung des Datenbankschemas erfordern.

Für FLEXOB wurde deshalb die Archivierung in strukturierten Textfiles realisiert. Um die Interpretierbarkeit bei der angestrebten Langlebigkeit zu gewährleisten, wurden die deskriptiven Daten, d.h. Objekt*beschreibungen*, deren Semantik aus der Domäne resultiert, von applikationsspezifischen Zusatzinformationen getrennt und in verschiedenen Files gespeichert:

- MODELL- BZW. PROJEKTFILES: Modellfiles (‘*.MOD’, persistente Klassen) und Projektfiles (‘*.PRJ’, konkrete Instanzen) enthalten die objektbeschreibenden Daten, wie Objektname, Attribute, Relationen, Subklassen, etc. Die Semantik der Fileeinträge ist durch die Anreicherung mit Schlüsselwörtern weitestgehend intuitiv erfaßbar. Die Filestruktur ist betont einfach und damit interpretationsfreundlich (ASCII-Zeichen, Trennung der Items durch Zeilenvorschub)
- SYMBOLFILES: Symbolfiles (‘*.SYM’, ‘*.SYP’) dienen zur Archivierung der Zuordnung der Objektidentifikatoren von Domänenobjekten zu den Identifikatoren der statischen C++-Klassen, die sie implementieren.

Die *Objektidentität* wird in der vorliegenden FLEXOB-Implementierung aus Effizienzgründen auf zwei Wegen realisiert:
- Identität durch Adressierung
- Identität durch Surrogate

FLEXOB verwendet für alle Objekte (Klassen, Instanzen, Attribute, Relationen, Facetten) einen 10 Byte langen, global eindeutigen (Surrogat-) Objektidentifikator. Für persistente Objekte, also Objekte, die in Modell- oder Projektfiles ausgelagert sind, werden zur Identifikation ausschließlich die zugeordneten Objektidentifikatoren verwendet.

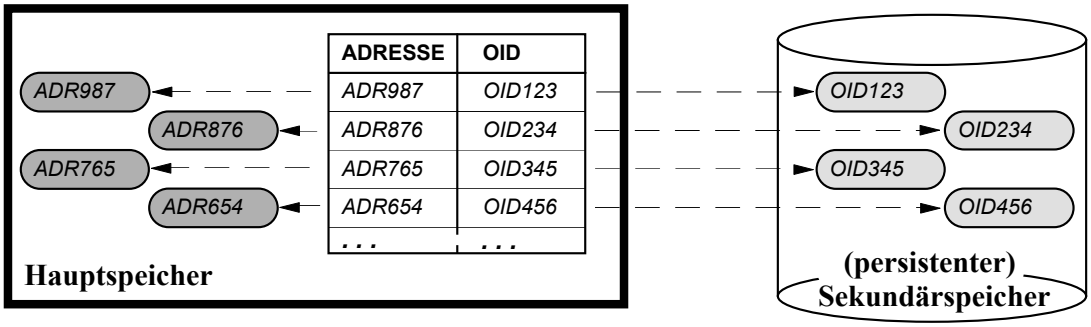


Abb. 8-15 Maping global uniquer Objektidentifikatoren zu Hauptspeicheradressen

Diese werden in der Phase des Einlesens in den Hauptspeicher mit Hilfe einer globalen Objekttabelle in Objektadressen übersetzt. Zur Laufzeit kann grundsätzlich wahlweise effizient

mit Mitteln der gewohnten direkten Adressierung gearbeitet werden oder über die Identifikatoren. Für Applikationen, die ihr Domänen- und Projektmodell vollständig im Hauptspeicher halten, wirkt sich jedoch der zweite Weg leistungsmindernd aus. (Abb. 8-15)

8.3.7 Nutzungstechnologie

FLEXOB ist durch Applikationsprogrammierer in zwei Varianten nutzbar als:

- C++ KLASSENBIBLIOTHEK: Der Applikationsprogrammierer bindet durch Linken die FLEXOB-Klassenbibliothek in seinen Code ein. Diese Variante kommt in Frage, wenn auch die Implementierung des Anwendungsprogrammes in der Sprache C++ erfolgt.
- OBJEKTSERVER: Diese Variante wurde insbesondere für die Nutzung in anderen Programmiersprachen geschaffen. Der FLEXOB-Objektverwaltungsmodul wird hier in der Form einer Windows-DLL verwendet. Dazu wird in einer Schnittstellenschicht ein Interface definiert, auf dem die Applikationsprogramme aufsetzen können (siehe Abb. 8-4 und 8-10 vorn). Es hat sich als günstig erwiesen, ein klassisches C-Funktioneninterface zu implementieren, da die meisten Sprachen keine C++-Klassen importieren können. Die Hauptaufgabe dieses Funktioneninterfaces liegt dann in der Umsetzung von Funktionsaufrufen in die C++-Methodenaufrufe (wrapperfunctions) der entsprechenden FLEXOB-Objekte. Diese Schnittstelle wurde erfolgreich zur Kopplung von FLEXOB mit einem in der Programmiersprache Delphi 2.0 implementierten graphischen Viewer (FLEXEDIT, [LAN96]) sowie zur Überwindung der „Sprachbarriere" zwischen C++-Systeme verschiedener Hersteller (Borland C++ 5.0, Microsoft C++ 4.0) eingesetzt.

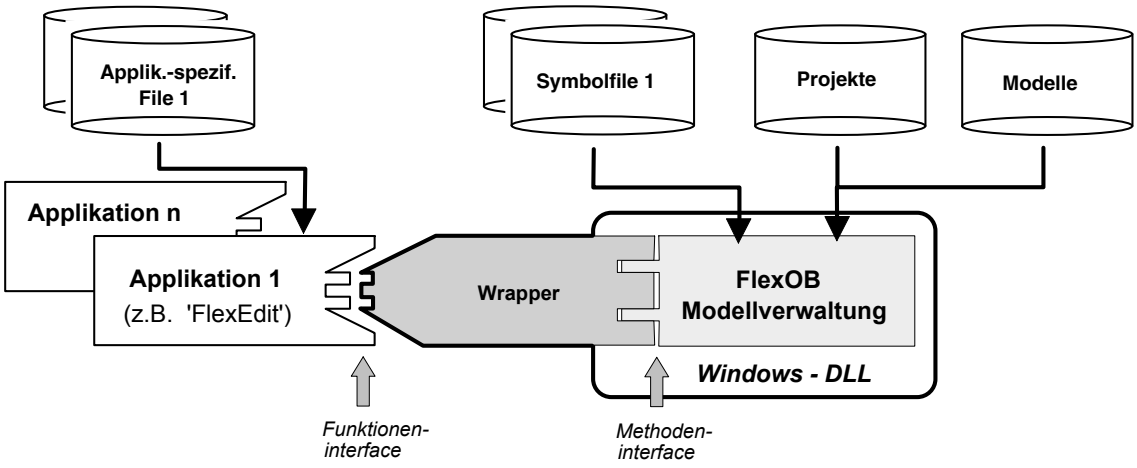


Abb. 8-16 Architektur der funktionalen Schnittstelle ‘AKO’ (siehe [KOL97])

8.3.8 Bewertung des Implementierungstools FLEXOB

Der mit Hilfe von FLEXOB implementierte generische Objekt- und Schemaeditor FLEXEDIT weist die Funktionsfähigkeit des Modellverwaltungskerns nach. In einem Testrahmen einer dynamischen Modellwelt für geometrische Objekte konnte auch für größere Objektmengen ein effizientes Laufzeitverhalten demonstriert werden (siehe Abb. 8-17 b).

Es bleibt zu bemerken, daß ein zusätzlicher Aufwand für Programmierung und Einarbeitung entsteht. Dies betrifft vor allem die Nutzungsphase I, der Anpassung des statischen FLEXOB-Kerns zur Applikationsentwicklung. Dies mag sich für Applikationen, die klein oder statisch programmierbar sind, nicht lohnen. Immer dann, wenn Anwendungsprogramme häufigen

Änderungen unterliegen oder kundenspezifisch adaptiert werden müssen, führt diese Technologie letztlich zu einer Reduktion des Aufwandes.

Testbeispiel: Das angeführte Testbeispiel entspricht in Objektmodell und Funktionalität dem ALOS Test aus Abschnitt 8.2.2 (Anhang F). Ein erweiterter Test, der insbesondere die Möglichkeiten der dynamsichen Klassifizierung und Beschreibung nutzt, enthält Anhang G.

A - 5000 Instanzen erzeugen und attributieren (*Konstruktion*)

B - Alle Instanzen entlang einer Relation zeichnen (reiner *Zugriff*)

C - Von ca. 50 % der Objekte entlang der Relation Massen berechnen (reiner *Zugriff*)

D - Alle Instanzen entfernen (*Destruktion*), Speicher freigeben

Testumgebung: Pentium100, 64 MB RAM, Windows NT 3.51

Getestet wurde der Modellierkern bis 25000 Instanzen, ab 15000 Instanzen setzte bei der verwendeten Rechnerkonfiguration Swapping durch das Betriebssystems ein, die Laufzeiten verlängerten sich dann stark.

Abb. 8-17a Generischer Schema- und Objekteditor FLEXEDIT

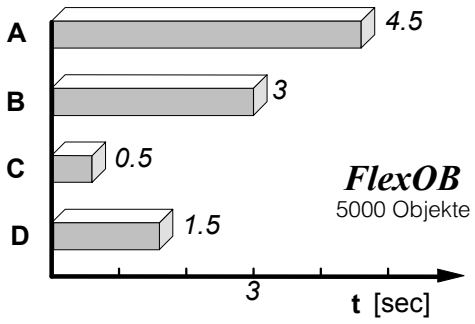


Abb. 8-17 b Laufzeittest für eine dynamische Modellwelt aus 2½D-Körper

9. FAZIT

CAD-Techniken werden in starkem Maße auf die Arbeitsumwelt des Architekten zurückwirken, sie verändern und fortentwickeln. Die Verwendung von Papier und Stift für die Externalisierung von Entwurfsideen bildet den historischen Ausgangspunkt für das planende Vorausdenken von Bauwerken, um diese dann kollektiv zu realisieren. Die Entwicklung von Repräsentationen für Entwurfsideen, die geeignet sind, diese Ideen anderen am Entwurf oder Fertigung beteiligten Personen mit hoher Authentizität zu vermitteln, sind die Basis der heute üblichen hochgradig arbeitsteiligen Planungs- und Fertigungstechnologien. Diese Arbeitsteilung bedingt jedoch einen immer höheren Organisations- und Kommunikationsaufwand, nicht nur beim Architekten. Will man auch weiterhin kreatives Arbeiten ermöglichen, müssen diese Aufgaben eine umfassende Unterstützung erfahren. Eine Unterstützung durch CAD-Technologie ruht auf zwei Säulen:

- *Adäquate Modelle*

  Der Architekt erzeugt durch die benutzte Software Modelle, die die imaginierte Realität mit nur geringem semantischen Abstand wiedergeben. Alle Aspekte und Entwicklungsstufen des Entwurfs sollten homogen unterstützbar sein. Dies gilt besonders für die speziell im Bauwerksentwurfs benötigte Analyse und Fixierung von Entwurfszielen. Die verwendete Modellierungsmethodik muß mächtig genug sein, um Modelle von einer Komplexität erzeugen und verwalten zu können, die dem Bauwerksentwurf in der Praxis gerecht wird.

- *Interpretierbare Modelle*

  Legt man für einen Kommunikationsprozeß das Kommunikationsmodell nach WINOGRAD ([WIN86]) zugrunde, so basiert jegliche Kommunikation auf einem Interpretationsprozeß, der durch die Kommunizierenden im Kontext ihres Wissen geleistet wird. Dieses Wissen ist an das Subjekt gebunden, das die Interpretation bewerkstelligt. Dies gilt letztlich auch für Software als Vergegenständlichung des Wissens, das durch die Entwickler zusammengetragen und ausgewählt wurde. Um eine monotone Interpretation zu ermöglichen, bieten sich zwei Wege. Dies ist zum einen die Normung von konsensfähigem Wissen, zum anderen der Transfer von Modellen und dem zu deren Interpretation nötigen Wissen. In der Praxis wird heute aus vielerlei Gründen der erste Weg bevorzugt. Für frühe Entwurfsphasen, wie dem angeführten funktionalen Bauwerksentwurf, existiert jedoch kein standardisiertes Domänenmodell. Der zweite Weg setzt im Rahmen des WINOGRAD'schen Kommunikationsmodells Wissen über ein Metamodell voraus, das homogen die Beschreibung von dynamischen Domänen- und Bauwerksmodellen erlaubt. Dieses Metamodell kann nun Gegenstand einer Normung sein.

Das vorgestellte PREPLAN-Konzept befaßt sich sowohl mit der Generalisierung eines Diskursbereiches als auch mit der Bauwerksmodellierung, d.h. dem Erzeugen von Abbildern realer oder vorgedachter Bauwerke. Diese Vorgehensweise wird möglich durch die Verwendung einer einheitlichen Modellierungsmethodik, der *Objektorientierung*. Mit ihr ist ein geeignetes Paradigma gegeben, das einerseits eine genügend hohe Modelliermächtigkeit besitzt und das andererseits als Metamodell bereits einen hohen Grad an Standardisierung erlangt hat. So können die traditionellen Ausdrucksmittel im Entwurf, wie Skizzen, Texte und Zeichnungen, um die Möglichkeiten erweitert werden, rechnerinterne Modelle zu bilden, die sich konsequent an ihren Urbildern orientieren. Der Einsatz solcher Modelle verspricht eine Systematisierung des Entwurfsprozesses als Ganzes und ermöglicht so in viel stärkerer

Weise, die spezifischen Fähigkeiten des Werkzeugs Computer sinnvoll in den Entwurfsprozeß einzubringen, als dies für reine Repräsentationen gelingt. Die Modellierungsmethodik basiert auf einer kognitiv begründeten Theorie, die, wenn auch nicht unumstritten, doch viele beobachtbare Phänomene erklärt bzw. beschreibt und die sich in Theorie und Praxis von Wissenschaft und Ingenieurwesen etabliert hat. Das Paradigma ist intuitiv verständlich und verknüpft in der integrierenden Einheit des Objektes beschreibendes Wissen bzw. Daten wie Attribute, Relationen, Dokumente, mit prozeduralem Wissen, wie Methoden oder Constraints. Alte Projekte (Modelle) können als Ausprägung des Domänenmodells in einen neuen Entwurfskontext assoziiert und an die konkreten Randbedingungen adaptiert werden.

Der für die CAD-Tool- bzw. -Systementwicklung notwendigen Formalisierung sind jedoch Grenzen gesetzt, die aus den Besonderheiten von Bauwerken als Entwurfsgegenstand einerseits und damit verbunden der des Planungsprozesses andererseits herrühren. Eine algorithmische und damit durch Computer unterstützbare Evaluierung bzw. Generierung von Bauwerksmodellen ist in den in der Arbeit behandelten frühen Phasen nur in reduziertem Umfang möglich. In diesem Prozeß rückt die Strukturierung und Verwahrung von Informationen sowie deren gezieltes Retrieval in den Mittelpunkt des Interesses. Dies ist von um so größerer Bedeutung, da sich die Erkenntnis durchsetzt, daß Bauwerksmodelle beträchtliche ökonomische Werte verkörpern und somit über die langen Zeiträume des Gebäudelebenszyklus verfügbar und unabhängig von der sie erzeugenden Software interpretierbar bleiben sollen.

Realistischerweise muß man aber sagen, daß man von den formulierten Zielen noch weit entfernt ist. Dies liegt vor allem in dem evolutionären Charakter der beobachteten Entwicklung begründet. Der in der Arbeit vorgestellte Ansatz eines nicht primär geometrischen, sondern semantischen Datenmodells, das die Geometrie lediglich als einen Aspekt beinhaltet, ist gegenwärtig nur im Bereich der Softwareentwickler populär. So resümiert auch DRACH für das Projekt A+ / Armilla, das sich wie FUNPLAN dem modellbasierten architektonischen Vorentwurf widmet: „Architekten sind gewohnt, in Zeichnungen zu denken und zu kommunizieren. Die von A+ angebotene Begriffswelt ... (beinhaltet) Abstraktionen, die in ihrer Anschaulichkeit und logischen Trennung ungewohnt sind.“ ([DRA94]). Die Potenz der vorgestellten Konzepte wird in der Praxis erst nutzbar werden, wenn spezifisch anpaßbare CAD-Tools diese Modellierweise umsetzen und in ihrer Abfolge durchgängige CAD-Systeme bilden. Diese Systeme entstehen *nicht* durch eine a priori geplante oder gar programmierte Sequenz der Toolnutzung, sondern durch die a posteriori beobachtbare, zweckspezifische Reihenfolge der Toolnutzung.

Hierdurch wird auch die weitere Forschungsstrategie auf diesem Gebiet deutlich. Die Menge der verfügbaren Tools muß erhöht werden, um zu demonstrieren, daß relevante Planungsprozesse durchgängig unterstützbar sind. Für die Programmierung von dynamisch modellbasierten Tools muß einerseits die Entwicklungstechnologie verbessert werden (etwa in Form eines Frameworks für das vorgestellte Modellverwaltungssystem FLEXOB) und andererseits muß eine Schnittstelle verfügbar sein, die einen Bauwerksmodell- und Domänenmodellaustausch auf der Basis des standardisierten Metamodells gestattet. Diese Schnittstelle löst den an sich obsoleten Datenaustausch via Dateien ab, wie er auch für FUNPLAN/ INFPLAN/ NETGEN Verwendung fand. Mit der im Rahmen des ‘ARBEITSKREISES OBJEKTE’ erarbeiteten Schnittstelle ([AKO97]) und deren FLEXOB-basierten Umsetzung ([BAC97]) liegen hierfür bereits Vorschläge vor.

Zum Abschluß dieser Arbeit sei noch einmal auf die philosophisch geprägten Erörterungen am Anfang verwiesen. Die vorherrschende Art der wissenschaftlichen Betrachtung und mit ihr auch diese Arbeit, geht vom repräsentationalistischen Ansatz aus. Er ist beim gegenwärtigen Stand der Softwaretechnologie für den Aufbau von computergestützten Werkzeugen essentiell. Man nimmt an, daß sich bei beliebig steigerbarem Aufwand eine beliebige Annäherung unserer Modelle an die Sachverhalte erreichen läßt, die sie abbilden. Diese Annahme ist bereits für beobachtbare Originale nicht unstrittig, da sie den Abbildungsaufwand außer acht läßt. Für den Entwurf, also der Vorabbildung eines Objekts, vernachlässigt sie darüber hinaus auch die Dynamik, mit der sich sowohl der repräsentierte Weltausschnitt als auch die Repräsentationsverfahren selbst ändern.

Abb. 9-1 M.C. ESCHER „Chaos und Ordnung“ 1950, in [ESC92]

Bei aller Zweckmäßigkeit der Bemühung um modellhafte Repräsentation eines Weltausschnitts sollte man sich sowohl der Lebendigkeit dieser Welt, als auch ihres fraktalen Charakters bewußt sein. Nur dann wird man nicht nur die Ergebnisse sondern auch Weg und Ziel der Bemühungen zur Verfügbarmachung digitaler Modelle immer wieder kritisch hinterfragen.

LITERATUR

[ABE95] Abeln, O.: „CAD-Referenzmodell“, Teubner Verlag, Stuttgart, 1995

[ACH95] Achard, G., Chevalier, G., u.a. : „Environmental design of building products: solutions to LCA boundary problems“, in "Critical Review of the Applications of Advanced Technologies", Proceedings of the 5.EuropIA - Lyon, Europia Productions, Paris,1995

[AKI86] Akin, Ö.: „Psychlogy of Architectural Design“, Pion Limited Publ., London, 1986

[AKI88] Akin, Ö.: „Expertise of the Architect“,
in: Rycher,M. (Ed.), "Expert Systems for Engineering Design",
Academic Press, London, 1988

[AKO97] ‘Arbeitskreis Objekte’ : „Schnittstelle AKO“, Stand 2/97,
verfügbar unter http://WWW.UNI-WEIMAR.DE/Bauing/biwv/akokurz.html

[ALE79] Alexander, C.: „The Timeless Way of Buildings“,
Oxford University Press, New York, 1979

[AND92] Anderl, R.: „STEP - Schritte zum Produktmodell“,
CAD/CAM Report, Nr. 8/92, Dressler Verlag, Heidelberg, 1992

[APP90] Apple Computer Inc.: „Human Interface Guidlines: The Apple Desktop Interface“, Addison-Wesley Publishing Company Inc., 1990

[ASI62] Asimow,M.: „Introduction to design",
Prentice Hall, Englewood Cliffs, New Jersey, 1962

[ATK90] Atkinson, M; Dittrich,K, u.a.: „The object-oriented databasesystem manifesto“, in Mitteilungsblatt GI-Fachgruppe 2.5.1 Datenbanken, 5/90

[AYE91] Ayerle, H.: „XNET2 - Methodical Design of Local Area Networks in Buildings: An Application of the A4 Intelligent Design Tool",
in "CAAD futures ’91“, Proceedings International Conforence for Computer Aided Architectural Design, Edited by G.N. Schmitt, Zürich, 1991

[BAB30] BABYLON „Referenzhandbuch“, Vers. 3.0, VWGedas, Berlin, 1993

[BAC97] Bachmann, U.: „Untersuchung zur Strukturierung einer Schnittstelle von CAD-Tools und dynamischen Modellverwaltungssystemen“, Diplomarbeit, Bauhaus Universität Weimar, Fertigstellung Juni 1997, Weimar, 1997

[BAK93] Bakhtari, S.; Bartsch-Spörl, B.; Hovestadt, L.: „Problemtypen und Problemlösungsmethoden : Typus Bau-Design“,
Fabelreport 18, Sank Augustin, 1993

[BAK94] Bakhtari, S.; Bartsch-Spörl, B.; Börner, K., Voß, A.: „Cased-Based Reasoning and Learning for Design“, Fabelreport 25, Sank Augustin, 1994

[BAR95] Barth,B.; Heintz,S. u.a. : „Life cycle modelling of buildings. Life cycle impact assesement and building specification“,
in "Critical Review of the Applications of Advanced Technologies",
Proceedings of the 5.EuropIA - Lyon, Europia Productions, Paris,1995

[BAY90] Baykan,C.A.; Fox, M.S. : „Constraint Satisfaction for Spatial Planning“,
in Hagen,P., Veerkamp, P. , : ‘Intelligent CAD Systems 3 - Practical Experience and Evaluation’, Springer, Eurographics Seminar, 1990

[BEC92] Becker, B. : „Künstliche Intelligenz : Konzepte, Systeme, Verheißungen“,
Campus Verlag, Frankfurt / M., 1992

[BEI85] Beitz, W : "Kreativität des Konstrukteurs",
Konstruktion 37/1985, S.381 - 386, Springer Verlag, Berlin 1985

[BER94] Bertzky,R. : „Anforderungen des Unternehmens an Facility Fanagement“,
in BauInformatik, 6/94, Verlag R.Müller, Köln

[BER95] Bergmann, A.; Bode, T., Cremers, A.B.; Reddig, W.: „Integration civil engineering applications with object-oriented database management systems“,
in Computing in Civil and Building Engineering“, Hrsg. J.P.Pahl,
Balkema Verlag, Rotterdam, 1995

[BON95] Bonlanger, S.,;Gelle, E.; Smith, I.: „Taking Advantage of Design Process Models“, in ’Konwledge Supported Systems in Civil Engineering’,
Report of the IABSE colloquium, Bergamo, 1995

[BOO94] Booch , G: "Object Oriented Analysis and Design",
Benjamin Cummings Publ., Redwood City CA, 1994

[BÖR95] Börner, K.: „Modules for Design Support“, in : „Knowledge based Support for architectural design“, Fabelreport 35, Sank Augustin , 1995

[BRA96] Braxein, S.: „Erstellen eines objektorientierten Bauwerksmodells zur ökologischen Langzeitbewertung von Gebäuden“,
Diplomarbeit, Bauhaus Uni-Weimar, Juni 1996

[BRE92] Breutmann, B.; Burkhardt, R.: „Objektorientierte Systeme“,
Hanser Verlag München, 1992

[BRO73] Broadbent,G.: „Design in architecture“, Wiley&Sons London, 1973

[BUN90] Bundy, A. : „Catalogue of Artificial Intelligence Techniques“, Springer Verlag, Berlin/ Tokio/ New York, 1990

[CAR94] Carrara,G.; Kalay, T.E.; Novembri, G.: “Knowledge based computational Support for architectural design“,
in: “Knowledge based Computer Aideded Architectural design“,
Elsevier, Amsterdam, Tokyo, New York, 1994

[CHA89] Brown,D.C; Chandrasekaran,B.: „Design Problem Solving",
Pitman London,1989

[COR92] „The Common Object Request Broker: Architecture and Specification“, OMG Document Nr. 91.12.1, Rev 1.1 (Draft), 1992

[COY90] Coyne,R.; Roseman, M.; Gero, J. et.al. : „Knowledge based design systems“, Addison Wesley, Reading, 1990

[CUN88] Cunnigham, J.J.;Dixon , J.R.: "Design with Features : The Origin of Features", in Computers in Engineering ed. V.A. Tipnis, E.A.Patton,pp.237-242, San Francisco, 1988

[CUN91] Cunis,R.;Günther,A.;Strecker,H.: "Das PLAKON-Buch", Informatik Fachberichte 266, Springer Verlag, Berlin, Heidelberg, New York , 1991

[CUN95] Cunis, R.: „KI-Programmierung in C++“, Kursscript, KI-Frühjahrsschule, Günne, Feb. 1995

[DAH96] Dahlenburg, A. : „Modellierung architektonischer Objekte in frühen Entwurfsphasen mittels architekturspezifischer Fachsprache“, Dissertation, Hochschule für Architektur und Bauwesen Weimar - Universität, Weimar, 1996

[DEN92] Denert, E.: „Software-Engineering“, Springer Verlag, Berlin/ Tokio/ New York, 1992

[DOM95] Domer, B. : „Integration vorhandener Software im Bereich der Tragwerksplanung“, in VDI Fortschrittsberichte Nr. 173, VDI Verlag, Düsseldorf, 1995

[DON88] Donath, D.: „Untersuchung zur anwendungsspezifischen Kommunikation und Modellierung im computergestützten Architekturentwurf“, Dissertation, Hochschule für Architektur und Bauwesen Weimar, 1988

[DON91] Dontah, D.; Maye, H.G.: „Untersuchung zum Stand des EDV-Einsatzes in Thüringer Architekturbüros“, in Deutsches Architektenblatt Ost, Nr. 10, 1991

[DON94] Donath, D.: „Funktionsorientiertes Entwerfen mit FUNPLAN", Materialien zur CBit ’94, Hochschule für Architektur und Bauwesen, Fakultät Architektur, Weimar, 1994

[DON95] Donath,D.; Regenbrecht, R.: „VRAD (Virtual Reality Aided Design) in the early Phases of Architectural Design Process“, in CAAD futures ‘95, Vol.1, PrePrint, Singapore, 1995

[DON96] Donath, D.; Ott, C. u.a. : „Projektstudie GebIS“, unveröffentlicht, Hochschule für Architektur und Bauwesen Weimar, 1996

[DOR91] Dorffner,G. : „Konnektionismus“, Teubner Verlag, Stuttgart, 1991

[DÖR91a] Dörner, H.: "Eine Begriffswelt und ein Rahmen des Konfigurierens", in ’Beiträge zum 5. Workshop Planen und Konfigurieren’, Hrsg. R.Cunis, Labor für KI-Forschung, Hamburg, 1991

[DÖR91b] Dörner, H.: „Modelle für das wissensbasierte Konfigurieren“, Habilitationsschrift, Universität Halle, 1991

[DÖR93] Dörner, H. in: Beiträge zum Workshop ’Modelle beim Konfigurieren’, LKI-M-93/3, S. 84-94 ,Labor für Künstliche Intelligenz, Universität Hamburg

[DOS90] Doster, A.; Werner, H.: „Objektorientierte Modellierung einer Wissensbasierten Benutzeroberfläche im konstruktivem Ingenieurbau", in "KI im Bauwesen" Hrsg.: J.Gauchel, Ernst & Sohn, Berlin, 1990

[DRA94] Drach, A.: „Flexible Werkzeuge für die integrierte Gebäudeplanung“, Dissertation, Uni. Karlsruhe, VDI-Berichte Reihe 4/ Nr. 125, VDI Verlag Düsseldorf, 1994

[DREY91] Dreyfus,H.L.; Dreyfus,S.E. : „Künstliche Intelligenz - Von den Grenzen der Denkmaschine und dem Wert der Intuition“, RoRoRo Verlag, Hamburg, 1991

[EBE92] Eberle, W. u.a.: “Alltag im Architekturbüro“, in AutoCAD Magazin special, iWT-Verlag Vaterstetten, 1992

[EHL93] Ehlers, H.: „Untersuchung funktionaler Kathegorien in Architektur und Stadtplanung“, Studienarbeit an der Stratclyde University Glasgow, 1993

[ENC90] Encarnacão, J.L.; Lindner, R.; Schlechtendahl, E.G: „Computer Aided Design“, Springer Verlag, Berlin/ New York, 1990

[ESC92] Escher, M.C. : „Graphik und Zeichnungen“, Verlag Benedikt-Taschen, Köln 1992

[EXP89] EXPRESS, ISO 10303-11/ Part1 draft: „The EXPRESS Language“, document CD 10303-11, ISO, Genf, 1989

[FIS94] Fischbach, R. : „Produktmodellierung im Bauwesen“, in BauInformatik, 6/94, Verlag R.Müller, Köln

[FLE90a] Flemming, U. : „Knowledge Representation and Acquisition in the LOOS System“, in ‘Building and Enviroment’ num. 25, page 209-219, 1990

[FLE90b] Flemming, U.: “Regelbasierte Systeme in der Architektur“, in: “Ki-Forschung im Baubereich“, Hrsg. Gauchel, J., Verlag Ernst & Sohn, Berlin, 1990

[FLE94] Flemming, U. : „Artificial Intelligence and Design: A Mid-term Review“, in Carrara, G.; Kalay, Y.E ed. : „Knowledge based Computer Aided Architectural Design“, Elsevier, Amsterdam, Tokyo, New York, 1994

[FLE96] Felmming, U. aktuell zu ‘SEED’ und Anwendungen auf http://logan.edrc.cmu.edu:8001/ACL_1/

[FOR97] Forgber, U.; Müller, C.: „A Planning Process Model for Computer Supported Cooperative Work in Building Construction“, in Tagungsband 15. IKM, Bauhaus Uni. Weimar, Feb.1997

[FRE85] French, M.: „Conceptual design for engineering“, Springer Verl., Berlin, 1985

[GAR92] Gardner, H.: „Dem Denken auf der Spur“, Klett-Cotta Verlag, Stuttgart, 1992

[GAR94] Garrett, J.H.: „Class-Centrered vs. Objectcentred Approaches for Modeling Engineering Design Information“,
in Tagungsband 14. IKM, HAB Weimar, März 1994

[GER89] Gero, J.S.; Roseman, M.A. „A conceptual Framework for knowledge-based Design at Sydney University’s Design Computing Unit“,
in Gero, J.S. (ed.) : „Artificial Intelligence in Design“,
Springer, Berlin/ New York, 1989

[GER90] Gero, J.S. : „Design Prototypes: A Knowledge Representation Schema for Design“, AI Magazine 11 (1990), Heft 4, S. 27-36, 1990

[GER95a] Gero, J.S.: „Recent advances in computational models of creative design“, in Computing in Civil and Building Engineering“,
Hrsg. J.P.Pahl, Balkema Verlag, Rotterdam, 1995

[GER95b] Gero, J.S.: „Computers and Creative Design“,
in CAAD futures ‘95, Vol.1, PrePrint, Singapore, 1995

[GI89] Gesellschaft für Informatik e. V.: "Referenzmodell für CAD-Systemen",
Fachaus.4.2.2: ‘Rechnergestütztes Entwerfen, Projektieren und Fertigen’,
Stand: 21. 2.1989

[GI91] Gesellschaft für Informatik e. V.: "Gestaltungsempfehlung für Benutzungsoberflächen von CAD-Systemen",
Fachaus.4.2, Fachgr.4.2.1, Stand: 3. 12.1991

[GI92] Gesell. f. Informatik, FG 4.2.1. „Rechnerunterstütztes Entwerfen und Konstruieren (CAD)“, AK 2 „Benutzungsoberflächen von CAD-Systemen“;
„Gestaltempfehlungen für Benutzeroberflächen von CAD-Systemen“,
Bonn, 1992

[GLO91] Glorr,P.A.; Streitz N.D. : „Hypertext und Hypermedia",
Springer Verlag, Berlin/ Heidelberg, 1991

[GÜN92] Günther, A.: „Flexible Kontrolle in Expertensystemen für Planungs- und Konfigurierungsaufgaben“, Dissertation Uni Hamburg 1991,
KI-Dissertationsreihe Nr.3, infix-Verlag, 1992

[GYA92] Gyalokay, A.: „Analyse des Ingenieurentwurfsprozeß von Spezialbauwerken im Hinblickauf die Realisierung wissensbasierter Entwurfsunterstützungen“,
Diplomarbeit, HAB Weimar, 1992

[GYA94] Gyalokay,A. :“Analyse des Entwurfsprozesses unter dem Aspekt der Gestaltung von Entwurfshilfsmitteln dargestellt am Beispiel der Entwicklung eineswissensbasierten Systems zum Entwurf von Trinkwasserbehältern“,
Dissertation, Hochschule für Architektur und Bauwesen, Weimar 1994

[HAA88] Haas, W. : „Datenaustausch bei CAD-Anwendungen im Bauwesen – Wunsch und Wirklichkeit“, VDI Bericht Nr.700.4, VDI Verlag, Düsseldorf, 1988

[HAA93] Haas, W. (Hrsg.) : „CAD-Datenaustausch-Knigge“, Springer Verlag, Berlin / Heidelberg, 1993

[HAA95] Haake, J.M., Thüring, M., Haasebrook, J.: „Hypermedia“, Kurs 5.4 der KIFS’95, Günne, 1995

[HAL85] Haller, F. : „ARMILLA - Ein Installationsmodell“, Institut für Industrielle Bauproduktion, Uni Karlsruhe, 1985

[HAR80] Hartmann, J.: „Entwerfen: Einführung in die wesentlichen Tätigkeiten eines Architekten“, Kohlhammer, Stuttgart, 1980

[HÄR89] Härder, T. : „Klassische Datenmodelle und Wissensrepräsentation“, in it 2/89, Oldenbourg Verlag, 1989

[HEC94] Heck, P.: „Objektorientierte Modellierung am Beispiel der Integration raum- und bauteilorientierter Daten in einem zentralen Objektmodell“, in VDI Fortschrittsberichte Nr. 131, Reihe 20, VDI Verlag, Düsseldorf, 1994

[HEI94] Heinecke, A. M. : „Auswirkungen des Objektorientierten Modellierens auf die Arbeit des Konstrukteurs und die Gestaltung von Benutzeroberflächen“, in Tagungsband 14. IKM, HAB Weimar, März 1994

[HEIN93] Heinecke, A. M. : „Objektorientiertes Modellieren - ein neues Paradigma für die Benutzung von CAD-Systemen?“, in "Wissenschaftliche Beiträge zur Informatik", Heft 1993, Technische Universität Dresden, Fakultät Informatik, 1993

[HEIS93] Heisserrman,J. , Woodbury, R : „Generating Languages of Solid Models“, in proceedings Second ACM / IEEE Conf., Cannada, Montreal, May 1993

[HEU89] Heuser, K.C.: “Innenarchitekt -Raumgestaltung“, Bd. 1: „Grundlagen, Gestaltungsregel und Gesetzmäßigkeiten“, 4. Aufl., Augustus Verlag, Augsburg, 1989

[HEU92] Heuer, A.: „Objektorientierte Datenbanken - Konzepte, Modelle, Systeme“, Addison Wesley GmbH, Bonn, 1992

[HOAI] „HOAI - Honorarordnung für Architekten und Ingenieure in der ab 1. Januar 1991gültigen Fassung", Bundesministerium für Wirtschaft und Bundesministerium für Justiz, Bundesanzeiger, Köln 1991

[HÖC76] Höckert, M., Schönfeld, G. : „SAUNA Planung, Konstruktion und Ausführung“, 2. Aufl., Verlag für Bauwesen, Berlin, 1976

[HOF92] Hofstadter, D.R. : „Gödel, Escher, Bach - ein endlos geflochtenes Band“, Klett-Cotta Verlag, München, 1992

[HOL96] Hollenstein, R.: „Fast wie ein Tor zum himmlischen Jerusalem“, in ‘Neue Züricher Zeitung’, S.57, 5.7.96

[HOV91] Hovestadt, L. : „A4 - A model for an extensive use of computers in architecture“, proceedings of Second International Workshop on Computer Building Representation for Integration, Aix-les-Bains, 1991

[HOV94] Hovestadt,L.: "A4 - Digitales Bauen - ein Modell für die weitgehende Computerunterstützung von Entwurf, Konstruktion und Betrieb von Gebäuden", Dissertation, Universität Karlsruhe, VDI-Berichte Reihe 20/ Nr. 120, VDI Verlag Düsseldorf, 1994

[HÜB95] Hübler,R. , Kolbe,P. , Steinmann, F. : „Wissensbasierte Computerstützung des früher Phasen des architektonischen Entwurfs, Teil I - Konzeption und Realisierung des Systems PrePlan“, in „Computer und Architektur - Computereinsatz in frühen Entwurfsphasen“, Wissenschaftliche Zeitschrift der HAB weimar, Heft 4/94, Weimar, 1994

[HUG91] Hughes, J., G.: „Objektorientierte Datenbanken“, Hanser Verlag München, Wien; Prentice Hall London, 1991

[HUP95] Hupfer, P. : „Modelling of fuzziness“ , in „Computing in Civil and Building Engineering“, Hrsg. J.P.Pahl, Balkema Verlag, Rotterdam, 1995

[ING92] Ingwersen, P.: „Informationretrieval Interaction", Taylor Graham, London, 1992

[INS88] „INSPEC CLASSIFICATION“, ‘A classification scheme for physics, electrotechnology, computers and control’, © The Institution of Electrical Egineers, 1988

[JOE76] Joedicke, J. : „Angewandte Entwurfsmethodik für Architekten", Karl Krämer Verlag Stuttgart, 1976

[JOE93] Joedicke, J. : „Entwerfen und Gestalten“, Karl Krämer Verlag Stuttgart, 1993

[KAH95] Kahlen, H.: „Bauwirtschaft und umfassendes Facility Management“, BauInformatik 6/95, Verlagsges. R.Müller, Köln, 1995

[KAI94] Kaiser, M.: „Testen und Bewerten von Speicherkonzepten für objektorientierte CAAD-Systeme“, Diplomarbeit, HAB Weimar, 1994

[KAP20] „KAPPA-PC Reference Manual“ und „KAPPA-PC Users Guide“, Version 2.0.7, IntelliCorp., Mountain View CA., 1992

[KAR90] Karbach,W. , Linster, M.: „Wissensaquisition für Expertensysteme“, Hanser Verlag, München, 1990

[KEE89] Keene, S.: „Object-Oriented Programming in CommonLisp“, Addison-Wesley Publishing Company Inc., Reading/ Mass., 1989

[KOL94] Kolbe,P. : "Grafische Spezifikation von Grundrissen auf der Basis taxonomischen Wissens" ,Diplomarbeit , HAB Weimar, 1994

[KOL93] Kolodner, J.: „Cased-Based Reasoning“, Morgan Kaufmann Publ., San Mateo/CA, 1993

[KOL97] Kolbe, P. Ranglack; D.; Steinmann, F.: „Ein Schnittstelle für dynamische Objektstrukturen für Entwurfsanwendungen“, in Tagungsband 15. IKM, Bauhaus-Uni. Weimar, Feb.1997

[KOP93] Kopisch, M.: „Spatial Relations for Configuration Tasks in General and for the cabin Layout of the Airbus A340 in Particular", in proceedings 6. IEA/AIE’93, Edinburgh, Gordon and Breach Science Publishers, 1993

[KRA83] Krause, R.: „Entwurfsgrundlagen für Entbindungsabteilungen und Neonatologische Abteilungen“, Dissertation, HAB Weimar, 1983

[KRE94] Kretzschmar, H u.a..: „Computergestützte Bauplanung“, Verlag für das Bauwesen, Berlin 1994

[KRU91] Kruse,H.; Mangold,R.; Mechler,B. : "Programmierung Neuronaler Netze", Addison Wesley, Bonn 1991

[KRU93] Kruse, R.; Gebhardt, J.; Klawonn, F. : „Fuzzy-Systeme“, Teubner Verlag, Stuttgart, 1993

[KUH91] Kuhlen, R. : „Hypertext - ein nichtlineares Medium zwischen Buch und Wissenschaft“, Springer Verlag, Berlin/Heidelberg, 1991

[KUN93] Kunze, U. :"Development of knowledge based Design Systems", in proceedings of 5.ICCCBE ’93, Anaheim (CA), 1993

[KUR96] Kurtz, P.: „Der neue Skeptizismus“, in „Der hundertste Affe“, von Randow, G. (Hrsg.), RoRoRo, Reinbeck, 1996

[LAA91] Laabs,A.: "Editierbare Visualisierungen physikalischer Größen", in VDI Fortschrittsberichte, Reihe 4 Nr. 107, , VDI Verlag, Düsseldorf, 1991

[LAN96] Langenhan, C.: ‘Untersuchung zur Konsistenzsicherung zwischen Modellobjekten und Implementationsobjekten am Beispiel eines Editors für Funktionsplangraphen, Diplomarbeit BauhausUni Weimar, Weimar, 1996

[LAS89] Laseau, P.: „Graphic Thinking for Architects and Designers“, 2.edition, Van Nostrand Reinhold, New York, 1989

[LIE93] Liebich, T.: "Wissensbasierter Architekturentwurf - von den Modellen des Entwurfs zu einer intelligenten Computerunterstützung", Dissertation, Hochschule für Architektur und Bauwesen Weimar - Universität, 1993

[LYS94] Lysek, R.: „ArchiTron - Ein anderes CAD-Representationsmodell“, in Tagungsband 14. IKM, HAB Weimar, März 1994

[MAH85] Maher, L.M.: „HIRISE and beyound : directions for expert systems in design“, in ‘Computer Aided Design’, Vol 17 / 9, butterworth&Co, 1985

[MAR84] March, L. : „The logic of design“, in Cross, N. (Hrsg.): „Development in design methodology", Wiley, London , 1984

[MAR94] Martin, J.; Odell, J.J.: „Object-Oriented Analysis and Design“, Prentice Hall, Englewood Cliffs/ N.J., 1994

[MAR90] Martinez,T.;Ritter,H.; Schulten,K.: "Neuronale Netze Einführung in die Neuroinformatik selbstorganisierender Netzwerke", Addison Wesley, Bonn 1990

[MAS95] Mastiaux, L.: „Objektorientierte, CAD-gestützte Konstruktion geschraubter Anschlüsse im Stahlhochbau“, Diplomarbeit, Ruhr-Universität Bochum, 1995

[MAT87] Maturana, H.R.; Varela, F.J. : „Der Baum der Erkenntnis , die biologischen Wurzeln der menschlichen Erkenntnis“, Goldmann Verlag, Bern / München, 1987

[MAT91] Mattos, N.M.; Desloch,S. u.a.: "Knowledge based approach to Intelligent CAD for Architectural design", in proceedings of 4.IEA-AIE, 1991

[MAV94] Maver,T.W. : „MultiMedia in architectural presentation“, im Tagungsband des 14.IKM, Weimar, 1994

[MCD82] McDermott, J. : "R1 : A Rule-Based Configurer of Computer Systems", in Artificial Intelligence, North Holland, 1982

[MEC94] Mechler, B.: „Intelligente Informationssysteme", Addison Wesley, Bonn, 1994

[MED95] Medjdoub, B. : „Towards aid in preliminary design in architecture: ARCHIPLAN“, in Computing in Civil and Building Engineering“, Hrsg. J.P.Pahl, Balkema Verlag, Rotterdam, 1995

[MIN75] Minsky, M. : „A framework for representing knowledge", in "Mind design", (J.Haugeland), MIT Press Cambridge MA, 1975

[MIT88] Mitschang, B.: „Ein Molekül-Atom-Datenmodell für Non-Standardan-wendungen.“, Informatikfachberichte Nr. 135, Springer Verl., Berlin, 1988

[MOR87] Mortensen,M;Jänicke,J. : "Planen und Entwerfen im Bauwesen“, Verlag für das Bauwesen, Berlin, 1987

[MSE94] Microsoft (R) Encarta, Copyright ©, 1994, Microsoft Corporation

[NED93] van Nederveen, S., u.a. „Information Models for Integrated Design“, in : "CAAD futures ’93“, Proceedings International Conforence for Computer Aided Architectural Design, Pitsburg , 1993

[NEG70] Negroponte,N. : "The architecture machine", MITPress, Massachusetts, London, 1970

[NEG95] Negroponte, N.: „Total Digital“, Bertelsmann Verlag, Augsburg, 1995

[NET1] aus http://WWW.tau.AC.iL/taunews/Botta2.GIF

[NEU51] Neufert, E.: “Bauentwurfslehre“,
14. Auflage, Druckhaus Tempfelhof, Berlin, 1951

[NEW58] Newell,A.; Simon, H. : „Heuristic Problem Solving: The Next Advance in Operation Research“, in proceedings of VI. Operation Research, Jan. 1958

[NIE92] Nielson, J.: „Hypertext und Hypermedia“,
Academic Press, Boston, San Diego, ... , Toronto, 1992

[OTT95] Ott, C. : „ Untersuchung zur Speicherung von und Nutzung von Regel- und Fallwissen in objektbasierten CAAD-Systemen“, Diplomarbeit,
Hochschule für Architektur und Bauwesen Weimar-Universität, Weimar, 1994

[PES80] Peschel, M.: „Ingenieurtechnische Entscheidungen, Modellbildung und Steuerung mit Hilfe der Polyotimierung“, Verlag für Technik, Berlin, 1980

[PET 94] Petrovic, i., K.: “On some issues of development of computer aided architectural desgin systems“,
in: “Knowledge based Computer Aided Architectural design“, Carrava, C.; Kalay; T.E. Hrsg.,Elsevier Publ. Comp., Amsterdam, London, Tokyo, 1994

[PHA77] Phal,G. ; Beitz.W.: „Konstruktionslehre“, Springer, Berlin/Heidelberg, 1977

[PUP91] Puppe, F.: "Einführung in Expertensysteme",
2. Auflage, Springer-Verlag ,Berlin/Heidelberg, 1991

[RAN92] Ranglack, D. : „Eine objektorientierte Datenverwaltung für flexible Konfiguration von rechnergestützten Prozessen der Bauwerksplanung“,
Dissertation, HAB Weimar 1992

[RAS94] Rasmussen, J et.a. : „Cognitive System engineering", ... ,1994

[RED95a] Reder,B. : „Modelling of attributive fuzziness in CAD-Systems“ ,
in „Computing in Civil and Building Engineering“,
Hrsg. J.P.Pahl, Balkema Verlag, Rotterdam, 1995

[RED95b] Reder,B. : „ Modellierung attributiver Unschärfe in CAD-Systemen“,
Dissertation, Hochschule für Architektur und Bauwesen Weimar - Universität, Weimar, 1995

[REI91] Reimer, U. : „Einführung in die Wissensrepräsentation“,
Teubner Verlag, Stuttgart, 1991

[RIC88] Richter, P. : „Entwicklung einer integrierten Informationsstruktur für relationale Datenbanken im Bauwesen", Dissertation, Gesamthochschule Kassel, 1988

[RIC88b] Richter, M.M. : „Prinzipien der Künstlichen Intelligenz, Leitfaden und Monographien der Informatik“, Teubner Verlag, Stuttgart, 1988

[RIT92] Rittel, H.W.J.:, „Planen, Entwerfen, Design“,
Kohlhammer Verlag, Stuttgart, 1992

[RIT93] Ritter,H. :Kursscript "Neuronale Informationsverarbeitung und Robotik",
KIFS 1993 , Günne/ Uni Bielefeld 1993

[ROT92] Rottke, E.: "Rechnerunterstützter Tragwerksentwurfs für Architekten",
in VDI Fortschrittsberichte Nr. 116, VDI Verlag, Düsseldorf, 1992

[ROW87] Rowe, P.G.:„Design Thinking", MIT Press Cambridge Massachusetts, 1987

[RUM91] Rumbaugh. R; Blaha, M. W; Premerlani, W.et. al.: "Object-Oriented Modelling and Design", Prentice-Hall, EngleWood Cliffs NJ, 1991

[SAG91] Sager, W. : „Objektorientierte Programmierung - Übersicht und Begriffe",
in: ComputerMagazin 7-8/91, S. 34-39, Basten Verlag Herzogenrath, 1991

[SAU92] Sauer, H.: „ Relationale Datenbanken, Theorie und Praxis“,
Addison Wesley, Bonn, 1992

[SCH90] Schmitt, G.; Chen-Cheng, C.: “Lernfähige Entwurfssimulatoren“,
in: “Ki-Forschung im Baubereich“, Hrsg. Gauchel, J.,
Verlag Ernst & Sohn, Berlin, 1990

[SCH92] Schönfeld, J. W. : "Gebäudelehre", Verlag W. Kohlhammer, Stuttgart, 1992

[SCH93] Schlegelmilch, S.: Heller, B. u.a. : „Evaluation hybrider Expertensystemtools“,
Reportreihe ‘Modularisierung wissensbasierter Systeme’,
Uni Bielefeld, Bielefeld 1993

[SCH94] Schult,T.J.: „CYC’s Fiction“, ct 5/94, Heise Verlag, Hannover 1994

[SCH95] Schmigalla, A.: „Ein laufzeitdynamisches Userinterface für modellbasierte Applikationen“,
in VDI Fortschrittsberichte Nr. 173, VDI Verlag, Düsseldorf, 1995

[SKE95] Sklenar, S.: „Integration hypermedialer Dokumente“,
Diplomarbeit, Hochschule für Architektur und Bauwesen Weimar, 1995

[STA65] Stachowiak, H.: „Denken und Erkennen in Kybernetischen Modellen“,
Springer Verlag, Wien, New York, 1965

[STA73] Stachowiak, H.: „Allgemeine Modelltheorie“, Springer Verlag, Wien, 1973

[STA92] Stark, M.: „Feature basierter Ansatz zur Integration im Bauwesen am Beispiel der Tragwerksplanung“,
in VDI-Fortschrittsberichte, Reihe 4, Nr. 116, VDI-Verlag, Düsseldorf,1992

[STE80] Steele, G.L.: "The Definition and Implementation of a Computer Programming Language Based on Constraints",
Ph.D. Thesis, AI Laboratory, MIT, Cambridge MA, 1980

[STE90] Steinmann,F.; Hübler,R.: 'Wissensbasierter Entwurf von Trägerlagen- ein Beitrag zur Automatisierung von Syntheseaufgaben',
in Wissenschaftliche Zeitung der Hochschule für Architektur und Bauwesen, Issue No.3, pp. 108-112, Weimar,1990

[STE93] Steinmaetz, R. : „Multimedia Technologie - Einführung und Grundlagen", Springer Verlag, Berlin, New York, Tokio, 1993

[STE94] Steinmann, F. : „Anwendung des Prinzips sensorischer Karten zur Visualisierung komplexer Beziehungsnetze",
Tagungsband 14. IKM, HAB Weimar, März 1994

[STE97] Steinmann, F.; Wehner, R., Hübler, R.: „FLEXOB - Entwicklungstool für dynamische, modellbasierte CAD-Systeme",
in Tagungsband 15. IKM, Bauhaus Uni. Weimar, Feb.1997

[STEP92] STEP, ISO 10303-1/Part1 draft: „STEP - Overview and fundamental principles", ISO, Genf, 1992

[STO80] Stoyan, H.: „LISP-Anwendungsgebiete, Grundbegriffe, Geschichte", Akademie Verlag, Berlin,1980

[STO91] Stoyan, H.: „Programmiermethoden der Künstlichen Intelligenz", Band 1+2, Springer Verlag, Berlin / Heidelberg, 1991

[STR91] Stroustrup, B.: „The C++ Programming Language", Addison Wesley, Reading Mass., 1991

[TAN91] Tank, W.: „Wissensbasiertes vs. assoziatives Konfigurieren", in Günther, A; Cunis, R.: Beiträge zum 5. Workshop „Planen und Konfigurieren", LKI-M 1/91, Uni Hamburg, 1991

[TAN93a] Tank, W. : „Wissenbasiertes Konfigurieren - Ein Überblick",
in: KI-Künstliche Intelligenz 3/1993, S. 7-10

[TOO95] TOOLBOOK Nutzerhandbuch, Version 3.0, Asymetrix Corp., 1995

[TSO95] Tsou, J.Y.; Lee, T. : „Architectural Information System for design and construction - Strategy and implementation", in Computing in Civil and Building Engineering", Hrsg. J.P.Pahl, Balkema Verlag, Rotterdam, 1995

[VAR90] Varela,F.: "Kognitionswissenschaft, Kognitionstechnik - eine Skizze aktueller Perspektiven", Shurkamp, Frankfurt/M, 1990

[VDI85] „VDI Richtlinie 2221: Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte", VDI-Verlag, Düsseldorf, 1985

[VIE93] Viegener, J.; Maurer, A. : „Ein Ansatz zur Dynamisierung von Thesauri in Informationssystemen",
in 'Nachrichten für Dokumentation', Vol. 44/93, S. 285-292, Frankfurt /M., 1993

[WEH95] Wehner, R.: „Implementation eines Modellierkerns für laufzeitdynamische Objektstrukturen in CAD-Produktmodellen“,
Diplomarbeit, Hochschule für Architektur und Bauwesen Weimar, 1995

[WEN90] Wenzel, D.: „Untersuchung des architektonischen Entwurfes im industriellen Wohnungsneubau unter dem Gesichtpunkt der Konzeption von Hilfsmitteln für das Entwerfen“,
Dissertation, Hochschule für Architektur und Bauwesen, Weimar, 1990

[WIN86] Winograd, T.; Flores, F.: „Understanding computer and cognition“,
Norwood, New York, Ablex Publ., 1986

[WOS91] Wossnig, P.; Behaneck,M.; u.a. : "Vom CAAD zum Bild : Architektur als fotorealistische Erlebniswelt", SOFT-TECH, Neustadt, 1991

[YOU93] Yourden, E.: „Die westliche Programmierkunst am Scheideweg“,
Prentice Hall, London, 1993

[ZIM92] Zimmermann,H.-J.; Altrock,C.v. : „Prinzipien und Anwendungspotentiale der Fuzzymengentheorie“, KI 4/92, S.6-12, Organ des Fachbereichs 1 der Gesell.f.Informatik, FBO Verlag, Baden-Baden 1994

[ZIM93] Zimmermann,H.-J (Hrsg.) : „Fuzzy Technologie - Prinzipien, Werkzeuge, Potentiale.“, VDI-Verlag, Düsseldorf, 1993

Warenzeichen/ Produktbezeichnungen

Allegro Common Lisp (Franz Inc.); AutoCAD, AutoLisp, AME (AutoDesk Inc.);
BABYLON (VW Gedas); Baudatenbank (Baudatenbank GmbH / Celle);
BOCAD (BOCAD Software GmbH/ Bochum); EXECL (MicroSoft Corp.);
KappaPC (IntelliCorp.); KOPERNICUS (Hochtief Software GmbH); LinkWay (IBM);
MSDOS, MS Windows, WindowsNT(MicroSoft Corp.);
NexpertObject (NEXUS GmbH); OBJECTIVE CAD (IBB GmbH/Kassel);
ObjectStore (Object Design Inc.); ONTOS (ONTOS Inc.);
PALLADIO (AcadGraph GmbH/ München);
PARADOX (Borland Inc.); SOM/ DSOM (IBM); ToolBook (Asymetrix);
tools++ (Rougue Wave Inc.); Visual **C,** MFC (MicroSoft Corp.); UNIX (Bell laboratories)

ABBILDUNGEN UND TABELLEN

Anhang

zur

Dissertation

Modellbildung und computergestütztes Modellieren in frühen Phasen des architektonischen Entwurfs

vorgelegt von

Dipl.-Ing. Frank Steinmann

Weimar, Mai 1997

Gliederung Anhang

Anhang A - GLOSSAR

Autor: Dipl.-Ing. F. Steinmann

Die angeführten Erklärungen versuchen die aktuelle Verwendung der Begriffe wiederzugeben, ein einheitlicher Sprachgebrauch liegt jedoch (noch) nicht vor. Der Glossar ist deshalb eher als Definition der Begriffsverwendung im Rahmen dieser Arbeit zu verstehen, als eine allgemeingültige Begriffserklärung.
Der Glossar stützt sich wesentlich auf die kognitiven Betrachtungen von MATURANA/ VARELA [MAT87], auf das Konzept der Frames nach MINSKY [MIN75], auf die Methodik der OO-Analyse und OO-Design (von Software) nach BOOCH [BOO94] bzw. RUMBAUGH [RUM91] sowie auf ATKINSON et.a. [ATK90], soweit OO-Datenbanktechnologien betroffen sind.
Probleme bereitet vor allem die verschiedene intendierte Bedeutung der Begriffe im Kontext von Programmiersprachen zum einen und bei der Bildung von Modellen der Welt als Kommunikationsmedium zwischen Personen oder Prozessen zum anderen.

Aggregation

Objekte (↓) lassen sich je nach Modellzweck als *ein* geschlossenes, homogenes Objekt betrachten oder aber als eine aus Einzelteilen aufgebautes Komplexobjekt (↓), wobei dies für seine Bestandteile wiederum gilt. Dies ist eine essentielle Betrachtungsweise der Welt, die die Wahl eines zweckmäßigen Abstraktionsniveaus für eine Modellbetrachtung von Instanzobjekten gestattet. Insbesondere in technischen Domänen ist Aggregierbarkeit eine grundlegende Eigenschaft von Objekten. Die Aggregation ist eine paradigmatische Relation (↓) Die Semantik von Aggregationsrelationen auf Klassenobjekten ist nicht endgültig definiert[1]. Die Aggregationsstruktur eines Objektes läßt sich als heterarchischer Graph (siehe auch Taxonomie ↓) darstellen. Synonyme Begriffe sind Kompositionshierarchie und Aufbaustruktur.

Atomare Designaktion

Dies sind Entwurfsaktionen, die nicht durch eine Folge anderer Entwurfsaktionen darstellbar sind. Atomare Designaktionen bilden in ihrer Gesamtheit das kognitive Prozeßmodell des Entwurfs ab, d.h. alle erkenntnisstheoretischen Aspekte des Entwurfsprozesses werden behandelt.

Atome

Atome sind Objekte (↓), die innerhalb eines Kontextes (↓) nicht in Teilobjekte zerlegt werden können. Ihr Abstraktionsgrad ist kennzeichenend für die Modellart, in der gerade entworfen wird. Atome der Entwurfsdomäne können nie Komplexobjekt (↓) sein. Beim Wechsel des Kontextes können Atome erst zu Simplexen (↓) und später auch zu Komplexen werden, indem sie durch Teilobjekte verfeinert werden.

[1] Varianten wären, daß Instanzobjekte zu einem komplexeren Instanzobjekt aggregierbar sind oder daß sie bei ihrer Erzeugung standardmäßig aggregiert werden.

Attribut

Ein Attribut ist eine Art von Slot (↓), der eine geschlossene, d.h. vollständige Beschreibung eines Merkmals eines Objektes trägt. Diese Beschreibung ist reflexiv, d.h. es gibt keine Referenzen auf andere Objekte der betrachteten Modellwelt. In Attributen erfolgt keine Beschreibung von Verhalten (siehe Methoden ↓) des Objektes. Da die Attributbeschreibung in sich durch Facetten strukturiert sein kann, werden sie von Attributobjekten realisiert, die Objekte des Metamodells sind. In Programmiersprachen werden sie als Membervariablen bezeichnet. Im Sinne der OOP enthalten sie Daten.

Bauwerksmodell

siehe Objektmodell (↓)

Behaviorismus

Es handelt sich um eine besonders in den USA verbreitete Richtung in Philosophie und Psychologie, die die Existenz abgrenzbarer mentaler Repräsentationen als Analogie zu externen, beobachtbarer Erscheinungen verneint. Sie erklärt (intelligentes) Verhalten als Relationen zwischen Umwelteinflüssen (stimuli) und beobachteten Handlungen (respons). Sie ist verwandt mit der Schule des Assoziativismus und des Funktionalismus. [MSE94]

CAD-System

Ein CAD-*System* versucht einen Entwicklungs- bzw. Konstruktionsprozeß eines Produkts ganzheitlich[2] zu unterstützen. Ein *System* ist ein einheitlich geordnetes Ganzes, das über einen Aufbau aus Bestandteilen verfügt. Die Bestandteile sind hier die spezialisierten Tools. Basis einer CAD-Systembildung sind heute gleichartig strukturierte Modelle. Das System selbst wird zweckorientiert konfiguriert, d.h. es handelt sich um ein wandelbares System. Der gegenwärtige Sprachgebrauch behandelt 'CAD-Tool' (↓) und 'CAD-System' oftmals synonym.

CAD-Tool / CAAD-Tool

Der Terminus '*CAD*' bezeichnet den Prozeß der Entwicklung bzw. der Konstruktion eines (meist technischen) Produkts, der durch entsprechende Software unterstützt wird. Der gegenwärtige Sprachgebrauch assoziiert mit CAD das Vorhandensein eines geometrischen Modells, das i.allg. grafisch repräsentierbar ist.
Der Terminus '*CAAD*' bezeichnet spezifische CAD-Tools zur Unterstützung der Planungstätigkeit im Bauwesen im allgemeinen und des architektonischen Entwerfens im besonderen. (siehe auch 'Kategorien von CAD-Tools' ↓)

[2] In der Praxis stellt besonders im Bauwesen die Forderung nach Ganzheitlichkeit ein Ziel dar, das gegenwärtig nicht erreichbar ist. Im Maschinenbau reicht die Prozeßkette bis zur Fertigung ('*CIM*').

Defaultwert

Defaultwerte sind Standardannahmen für Wertausprägungen, d.h. für die Wertfacette (↓) dieses Merkmals. Sie können durch aktuelle(re) Werte ersetzt werden, wenn diese verfügbar sind. Dies gilt insbesondere für die Ausprägung von Merkmalen, die in abstrakten Klassenobjekten gültig sind und die durch Vererbung für konkrete Instanzen Gültigkeit erlangen. Defaultwerte können auch in speziellen Facetten abgebildet werden.

Dokument

Ein Dokument ist eine geschlossene Menge von Informationen. Ein Dokumententyp gibt eine Strukturierung dieser Informationen lediglich nach syntaktischen Gesichtpunkten vor. Die erforderliche Syntax wird durch eine Applikation bestimmt, mit der das Dokument repräsentiert bzw. bearbeitet werden kann. Die Repräsentation kann sich verschiedener Medien bedienen. Ein Dokument kann rekursiv aus Dokumenten aufgebaut sein.

Domäne

Die Gesamtheit von Modellen, die einem gewissen Blickwinkel auf die Welt oder einen überschaubaren Ausschnitt aus der Welt und deren Zustand zugeordnet sind, bezeichnet man als Domäne. Im Rahmen dieser Arbeit bezeichnet der Begriff ein Arbeits- oder Wissensgebiet, das durch seinen Inhalt und/oder seine Verfahren abgegrenzt ist.

Domänenmodell

Es handelt sich um ein verallgemeinertes Modell durch Abstraktion, das für eine Menge potentieller Ausprägungen gültig ist. Beschrieben werden Merkmale, anhand deren ein konkreter Sachverhalt diesem Domänenmodell zuordenbar ist, oder die für eine potentielle Ausprägung Gültigkeit erlangen werden. Objektmodelle (↓) und Domänenmodell verhalten sich zueinander wie Element und Menge.

Facette

Eine Facette ist eine weitere Untersetzung attributiver oder relationaler Beschreibungen in Slotobjekten (Attribut des Slotobjekts). Ihre Semantik und damit Verwendung ist nicht standardisiert[3]. Facetten bilden die terminalen Beschreibungselemente und enthalten in diesem Sinne Daten. OO-Programmiersprachen bieten keine Facettenkonzepte.

Focus

Bei der Bearbeitung von Entwurfsaufgaben wird i. allg. nicht ständig der Wirkungsort (Stelle im Aggregationsgraphen ↑) von Entwurfsoperationen gewechselt, sondern eine Sequenz von Entwurfsoperationen richtet sich auf ein bestimmtes Objekt. Dieser Wirkungsort von Entwurfshandlungen ist der **Focus**. Ein Wechsel erfolgt weiterhin meist zu im Strukturgraphen benachbarten Objekten (Lokalitätsprinzip). Der Focus dient dem Zweck, bei Entwurfsaktionen (z.B. Aggregieren oder Parametrisieren) nicht jeweils den Zielpunkt der Aktion spezifizieren zu müssen. So wird das Verhalten des Nutzers simuliert, der ‘ja weiß woran er gerade arbeitet’.

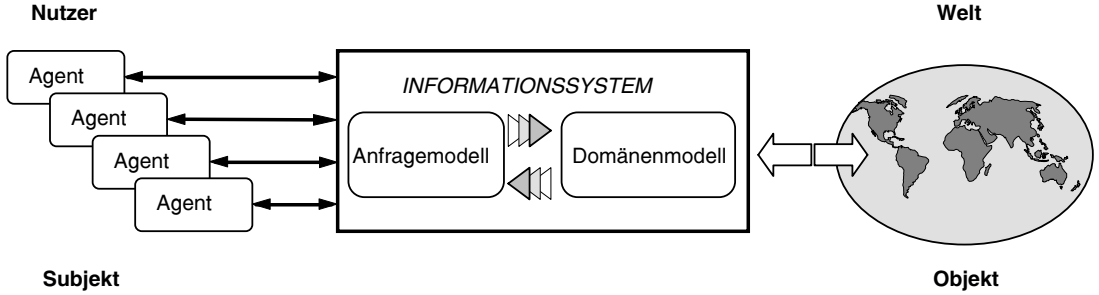
[3] Varianten wären kontextabhängige Wertveränderung von Slotobjekten, Überwachung von Slotobjekten, weiter Beschreibung von Slotobjekten u.a.

Hypermedial~

Eine Informationensammlung, die sich mehrerer Medientypen als Träger bzw. Mittler bedient (*multimedial)* und die ein komplexes Beziehungsnetz zwischen Einzelinformationen unterhält wird als hypermedial bezeichnet. Dieses Beziehungsnetz (Links) kann zur Erzeugung (Authoring) und Nutzung (Retrieval) verwendet werden. Eine solche Informationssammlung wird nichtsequentiell (d.h. nicht wie ein Buch) erzeugt und rezipiert.

Informationssystem (IS)

Der Begriff des Informationssystems stammt aus dem Bereich von Systemen zur Dokumentenverwaltung und -recherche. Die Systeme bieten in diesem Bereich Retrievaltechniken in schwach strukturierten Datenbeständen. Definitionen sind etwa zu finden in [ING92], [MEC94], [RAS94]. IS werden definiert als Systeme, die einen Mittler zwischen Akteur (Agent) und zu repräsentierenden Weltausschnitt realisieren. Agenten können hierbei sowohl Nutzer als auch Prozesse sein. IS stellen somit eine *operationale* und inhaltliche Erweiterung von Produktmodellen dar.



Instanz

Instanzen sind konkrete Ausprägungen von Klassen (↓). Die Bezeichnung Instanzobjekt wäre präziser. Im programmiersprachlichen Sinne sind Instanzen abgrenzbare Speicherbereiche, die einer Strukturierungsvorschrift genügen. *Instanzen sind konkret.*
(siehe 'real-world' Objekte ↓).

Kategorien von CAD-Tools

Die Klassifikation von CAD-Tools (↑) kann sich inhaltlich am unterstützten Gegenstandsbereich orientieren (domänenspezifisch, phasenspezifisch). Die folgende Kategorisierung richtet sich jedoch nach dem verwendeten Modelltypus (spezifisch oder generisch), beziehungsweise danach, ob und wie Modellausprägungen erzeugt, verändert oder ausgewertet werden. Eine eindeutige Zuordenbarkeit zu den Kategorien ist i.allg. nicht gegeben.

'representation and calculation aids'

Dies sind generische Entwurfs-*hilfsmittel*, die über keine spezifischen Produktmodelle verfügen. Sie ver- oder bearbeiten abstrakte Elemente ohne anwendungsspezifische Semantik, wie Linien, Flächen, Körper, Matrizen, Zahlen oder Strings. Beispiele wären Programme zur Textverarbeitung, Tabellenkalkulation, zeichnerischen Konstruktion, fotorealistischen Visualisierung, allgemeine Statistik- oder Optimierungsprogramme.

‘design assistant’

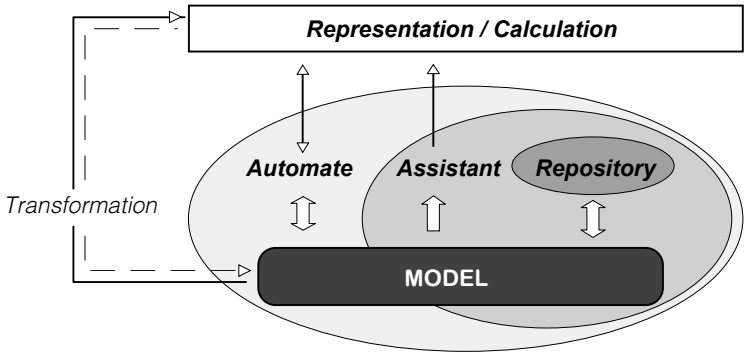
Diese Tools verfügen über ein Produktmodell der behandelten Domäne. Sie unterstützen den Nutzer beim Aufbau konkreter Bauwerksmodelle als Instanzen solcher Domänenmodelle. Der Umfang und die Variabilität der Domänenmodelle variiert dabei jedoch stark. Derartige Tools liefern Funktionen zur Plausibilitäts- und Konsistenzsicherung von Modellen sowie zur Evaluierung von Modellen. Der Aufbau der Modelle selbst bzw. die Selektion von Modellvarianten, erfolgt nutzergetrieben.

‘design automate’

Derartige Tools erweitern die Funktionalität eines ‘design assistant’ um Komponenten zum Generieren oder Variieren von Modellelementen. Soweit es sich nicht um spezifische Optimierungssysteme handelt, verwenden die Tools implizit oder explizit Wissen zur Vorgehensplanung, zur Variantenselektion und ansatzweise zum Repräsentieren von Entwurfskontexten. Auf dem Gebiet des CAAD sind allgemeine Kriterien der Wirtschaftlichkeit, Nutzbarkeit oder formale ästhetische Gesetzmäßigkeiten Grundlage für die Entscheidung. In der breiten Nutzung befinden sich gegenwärtig keine derartigen Systeme.

‘design repository’

Tools der Kategorien ‘design assistant’ bzw. ‘design automate’ arbeiten (heute) über fixierte Domänenwelten, d.h. der Vorrat an Beschreibungselementen zum Erzeugen und Bewerten von Planungsobjekten ist beschränkt. Bei einer Trennung von Modell und Algorithmus lassen sich jedoch auch neue Arten (Klassen) von Planungsobjekten erzeugen, die die Beschreibungsmächtigkeit des Domänenmodells erweitern oder anpassen. Ein ‘design repository’ organisiert das simple Erzeugen, Editieren und Verwahren derartiger Modellbeschreibungen. Es verzichtet auf Berechenbarkeit innerhalb der Domänenwelt. Basis eines solchen Systems ist eine Strukturierungsvorschrift für Modellwelten, d.h. ein Modellierungsparadigma. Derartige Programme gestatten ein standardisiertes Modellretrieval und sichern damit eine langfristige Interpretierbarkeit der Modelle. Sie können als Kern eines CAD-Systems dienen, in dem dann Tools der anderen Kategorien über dem Modell arbeiten.



Klasse

Klassen sind *abstrakte* Objekte. Sie haben kein direktes Analogon in einer modellierten Welt. In ihnen werden Eigenschaften definiert, die für eine Vielzahl von konkreten Ausprägungen Gültigkeit erlangen. Dies bezieht sich auch auf die mit einer Ausprägung (Instanz↑) assoziierten Informationen, die gegenwärtig nicht beobachtet werden (können) oder die nicht explizit angegeben worden sind. Das Geschlossenheitsmerkmal der Objekte (↓) gilt auch für Klassen, die Bezeichnung Klassenobjekt wäre somit präziser. Im programmiersprachlichen Sinne sind Klassen Strukturierungsvorschriften für abgrenzbare Speicherbereiche und die Zuordnung von Zugriffs- und Interaktionsfunktionen in Form von Methoden. In der OO-Programmierung wird von Klassen als Typen gesprochen, während sie aus mathematischer Sicht als eine Vorschrift zur Mengenbildung verstanden werden können.

Komplexe

Komplexe sind Instanzobjekte (↑), die über eine Aufbaustruktur (Aggregation↑) verfügen, d.h. sie verfügen über mindestens ein Objekt, das Teil des Komplexes ist. Diese Teilobjekte können Komplexe oder Simplexe (↓) sein. Komplexe können selbst Teil von etwas sein. Sie stellen die Nichtblattknoten der jeweiligen heterarchischen Aufbaustruktur dar. *Topkomplexe* sind die 'Spitzen' der heterarchischen Aggregationspyramide, d.h. sie sind selbst nicht in andere Komplexobjekte eingebaut.

Kontext

Der Kontext bezeichnet alle Informationen außerhalb eines Objektes, die dessen Semantik festlegen. Neben dem situativen Kontext als Gesamtheit aller aktuellen Projektinformationen ist dies insbesondere Wissen (beispielsweise in Form von Klassen). Ein Wechsel des Wissensausschnitts bei der Interpretation eines Objekts ändert seine Bedeutung, wobei sich sowohl die Art als auch die Menge von Informationen, die das Objekt trägt, ändern kann (siehe Sichten ↓).
Bemerkung: Im Sinne der OO-Programmierung wird ein Kontext durch ein Typsystem definiert, in dem das Objekt klassifiziert wird. Mit dem Begriff des Kontextes ist hier also in besonderem Maß das Verhalten von Objekten verknüpft.

Medium

Ein Medium ist ein Mittel zum Transport bzw. zur Verwahrung und Darstellung von an sich immateriellen Informationen. Es lassen sich dynamische, d.h. zeitabhängige Medien wie Sounds oder Videos und zeitunabhängige Medien, wie Texte oder (Vektor-) Grafiken unterscheiden. (siehe [STE93])

Meta~

Diese Vorsilbe bezeichnet lt. Fremdwörterbuch etwas "jenseits oder außerhalb stehendes". Gemeint ist die Beschreibung einer Beschreibung an sich, ohne Beachtung ihrer konkreten Bedeutung in einem Modellzusammenhang. Die Beschreibung an sich (d.h. deren Zeichen), steht außerhalb des Modells[4] (↓).

[4] Die Zwangsläufigkeit derartiger Metatheorien folgt aus dem Gödelschen Unvollständigkeitssatz, der besagt, daß zur Erklärung (der Widerspruchsfreiheit) einer Theorie immer auf Aussagen außerhalb der Theorie Bezug genommen werden muß.

Metaklasse

Dies ist eine Klasse (↑) eines Metamodells, die die Verwahrung von Klassenbeschreibungen eines Domänenmodells erlaubt. Die Ausprägung dieser Klasse erlaubt die Darstellung von Klassen an sich.

Metamodell

Ein Metamodell ist eine Menge von Ausdrucksmitteln, die eine erwünscht große Menge von Domänenmodellen explizit abzubilden vermag und damit implizit auch deren Ausprägungen. Die zur Darstellung von Domänen- und Objektmodellen verwendeten Ausdrucksmittel sind selbst nicht Teil dieser Modelle und müssen außerhalb dieser erklärt sein.

Metaobjekt

Ein Metaobjekt ist die Beschreibung des Objekts (↓) an sich. Sie wird gebildet als Ausprägung der Klasse (↑) eines Metamodells (↑), die die Verwahrung von Objektbeschreibungen erlaubt.

Metaschema

Ein Metaschema ist die Beschreibung eines Schemas an sich, ohne Betrachtungen deren Semantik in der Domäne. Metaschemata werden gebildet aus Beschreibungselementen des Metamodells.

Metaslot

Ein Metaslot ist die Beschreibung des Slots (↓) an sich. Er wird gebildet als Ausprägung der Klasse eines Metamodells (↑), die die Verwahrung von Slotbeschreibungen erlaubt.

Methode

Methoden sind eine Art von Slot (↓), die die Beschreibung eines bestimmten Verhaltens des Objektes (↓) beinhalten, dem sie zugeordnet sind. Sie sind kein eigentlicher Bestandteil einer deklarativen Modellwelt, sondern deren prozedurale, anweisungsorientierte Ausprägung. Das Verhalten eines Objeks wird durch die Gesamtheit seiner Methoden, aber auch durch seinen Modellzustand bestimmt. Die Verwendung von Methoden bedeutet letztlich immer das Organisieren von Prozessen. Ausgelöst werden diese Prozesse durch das Übermitteln einer *Nachricht* (Methodenaufruf) an ein Objekt, die aus dem Namen der Methode und Parametern besteht. Insbesondere in compilierenden Programmiersprachen sind Methoden nur mit Klassen (↑) assoziiert. Für Instanzen (↑)erlangen sie dann auf dem Weg der Vererbung Gültigkeit. Welche Methode bei welchem Objekt durch die Nachricht letztlich getriggert wird, wird zur Laufzeit entschieden (late binding). Die Reaktion verschiedener Objekte auf dieselbe Nachricht kann verschieden sein (Polymorphismus). Das Nachrichtenkonzept kann ganz im Sinne der Datenkapselung und der seiteneffektfreien Programmierung homogen auch zum Zugriff auf Slots benutzt werden, da nicht wesentlich ist, ob der Rückgabewert auf eine Nachricht durch einen Prozeß (Methode) oder durch simple Rückgabe des Slotobjektes zustande kam.

Modell

Modelle sind verkürzte Abbildungen eines zu betrachten Sachverhaltes. Die Art der Verkürzung oder Abstraktion wird vom Modellzweck bestimmt sowie durch den oder die Prozesse oder Personen, die diese Abbildung leisten. Sie sind mithin subjektv. Die Urbilder der Abbildung können existieren, aber auch imaginiert sein. Dieser Modellbegriff unterscheidet sich von generalisierten Modellen, die Mengen realer oder zukünftiger Ausprägungen beschreiben (siehe Domänenmodell ↑).

Modelleditoren

Modelleditoren stellen die Funktionalität zur Verfügung, um generisch (d.h. nur mit Kenntnis des Metamodells) den Aufbau, die Wartung und das Retrieval eines konkreten Modells über ein Modellverwaltungsystem (↓) zu ermöglichen.

Modellverwaltungssystem (MVS)

Modellverwaltungssysteme organisieren die Strukturierung und Verwahrung (sowohl transient als auch persistent) von Modellen. Wenn MVS auf Elementen eines Domänenmodells (Klassen ↑) basieren, dienen sie der Verwaltung von Domänenmodellausprägungen (von Instanz ↑). Wenn sie dagegen auf Metamodellelementen basieren, sind sie frei von Anwendungssemantik und können sowohl verschiedene Domänenmodelle als auch deren konkrete Ausprägungen verwahren. Sie stellen Transaktionen bereit.

Nachrichten

siehe Methoden ↑

Objekt

Ein Objekt ist ein Gegenstand oder Sachverhalt, der über Eigenschaften verfügt, die ihn klar aus einer Umgebung herausheben oder von einem Hintergrund trennen, d.h. er besitzt eine Identität. Für ein Objekt läßt sich ein ‘Innen’ und ‘Außen’ in bezug auf dieses Objekt definieren. Diese ‘Geschlossenheit’ ist wichtigstes Merkmal von Objekten.
Bemerkung: Die Nutzung des Begriffs Objekt ist in der OO-Softwaretechnik uneinheitlich. Inhaltlich richtig ist die Nutzung des Begriffes für Instanzen, da nur diese im eigentlichen Sinne über Identität verfügen (nur sie sind konkret). Da sowohl die externe syntaktische Darstellung als auch die interne Realisierung oftmals gleich sind, wird der Begriff Objekt häufig als Oberbegriff für *Klassen* und *Instanzen* verwendet. Konsequenterweise sollte aber von *Objektklassen* und *Objektinstanzen* gesprochen werden. (siehe ‘real-world’ Objekte ↓)

Objektidentifikator

siehe Objektidentität ↓

Objektidentität

Jedes Objekt (↑) verfügt über ein oder mehrere Merkmale, die dieses in der jeweiligen Modellwelt einmalig und von anderen Objekten unterscheidbar machen. Der Name des Objektes ist hierzu im allgemeinen nicht ausreichend. Bei Bedarf kann ein einzelnes Merkmal (Objektidentifikator) konstruiert werden, das diese Unterscheidbarkeit gewährleistet. Diese Unterscheidbarkeit sollte weder zeitlich noch örtlich beschränkt sein.

Objektmodell

Dies bezeichnet ein Modell eines konkreten Sachverhalts. Dieser Sachverhalt ist selbst beobachtbar ('real-world object' ↓) oder zumindest potentiell beobachtbar (Planungsobjekte). Das Objektmodell gibt den Zustand dieses Sachverhaltes zu einem fixierten Zeitpunkt wieder.

Objektorientiert (OO~)

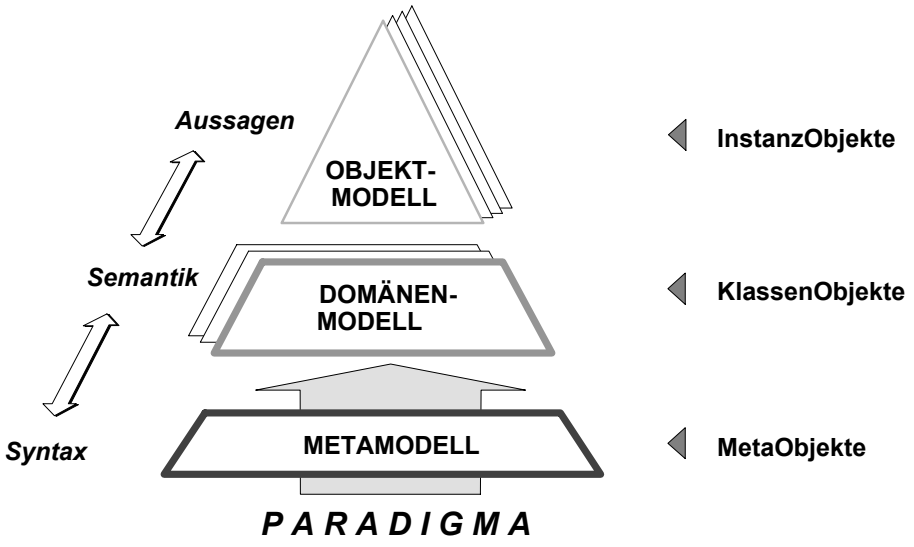
Objektorientiertheit ist die natürliche Sichtweise auf die Welt. Sie behandelt die Eigenschaften und das Verhalten von Objekten sowie Interaktionen zwischen Objekten. Objektorientiert heißt ganz wörtlich: *auf das Ding gerichtet !*

Objektorientiertes Modellieren (OOM)

Ausgehend von der Definition der OO ist OOM das schrittweise Erstellen von Modellen, d.h. entsprechend des Modellzwecks abstrahierende Darstellung realer oder vorausgedachter Umwelt durch *Klassifizieren*, *Benennen, Beschreiben* und *Konfigurieren* von Objekten sowie deren Beziehungen und Verhalten.[5] Zu unterscheiden ist wiederum das Modellieren von generalisierten Modellen bzw. das Modellieren als Erzeugen von Ausprägungen generalisierter Modelle.

Paradigma

Ein Denk- oder Betrachtungsmodell, d.h. die Art und Weise der Strukturierung von Objekt- und Domänenwelten.



Paradigmatische Relation

Relationen (↓), deren Semantik durch das verwendete Modellierungsparadigma (Paradigma↑) fixiert sind. Im Rahmen des OO-Modellierens sind dies 'Generalisierung/ Spezialisierung', 'Instanziierung/ Klassifizierung', 'Teil_von/Besteht_aus' (siehe Super-/Subklassen ↓, Aggregation↑).

[5] In [DOS90] wird ausgeführt: „Es liegt ... nahe, bei der Modellierung eines Problems die Realität direkt in ein objektorientiertes Modell abzubilden. Die Eigenschaften einer objektorientierten Betrachtungsweise, wie die Gliederung der Objekte in eine Klassenhierarchie und Vererbung von Eigenschaften, erlauben einen strukturierten und modularen Aufbau und eine schrittweise Verfeinerung des Modells."

Produktmodell

Siehe Domänenmodelle ↑

Projekt

Ein Projekt ist die Gesamtheit aller zu einem Zeitpunkt vorhandenen Instanzobjekte in all ihren Ausprägungen (entsprechend der jeweiligen Sichten). Es handelt sich also um alle mit einem Projektnamen assoziierten Informationen.

Projektmodell

siehe Objektmodell

Propagation

Gerichteter Prozeß, in dem Objekte erzeugt, gelöscht bzw. Merkmalsausprägungen erstellt oder verändert werden. Die formalisierten Verfahren der Propagation, die Trigger- und Terminierungsbedingungen dieses Prozesses sind Teil des Domänenmodells. Vererbung kann durch einen solchen Prozeß realisiert sein, ebenso wie andere Formen von Eigenschaftspropagation in hierarchischen Strukturen[6].

'real-world' Objekte

Dies sind Objekte (↑) eines Modellierungsprozesses, für die beobachtbare oder potentiell beobachtbare Urbilder für die modellhafte Abbildung existieren. Es handelt sich um Objekte, die in einer Anwendungsdomäne eine Semantik tragen bzw. eine Funktion erfüllen (im Gegensatz zur programmiertechnischen Objekten). 'real-world' Objekte sind Instanzen.

Relation

Relationen sind Beziehungen zwischen Objekten (↑). Sie sind in einer Art von Slot abbildbar, der dann eine transitive, d.h. referenzierende Beschreibung für ein Objektmerkmal enthält. Referenziert werden andere Objekte der jeweiligen Modellwelt. Da die Relationenbeschreibung in sich durch Facetten (↑) strukturiert sein kann, werden sie von Relationenobjekten realisiert, die Objekte des Metamodells (↑) sind. In Programmiersprachen werden sie in Membervariablen realisiert, die Referenzen auf andere Objekte enthalten (siehe auch Paradigmatische Relationen ↑).

Repräsentationalismus

Es handelt sich um eine traditionelle Schule der Philosophie, die von der Existenz von abgrenzbarer, mentaler Repräsentationen ausgeht (intrinsische Modelle). Für die Erklärungen (intelligenten) Verhaltens wird von einer Art von Verrechnungsprozeß (Räsonieren) über derartige Modellwelten ausgegangen [GAR92].

[6] Eine Variante wäre die Vermittlung von Wertausprägungen in der Aggregationsstruktur vom Komplex zu seinen Teilen (keine Form von Vererbung !).

Schema

Schemata sind die oberste Stufe geschlossener Beschreibungen innerhalb einer Domäne. Sie werden durch ein *System* von Klassen gebildet (Taxonomien ↓). Ein Schema bildet einen Aspekt einer komplexen Domänenwelt ab, der (weitgehend) unabhängig, d.h. von anderen Aspekten entkoppelbar ist.

Sicht

Eine Sicht ist ein Partialmodell, d.h. ein Ausschnitt des Gesamtprojektes (siehe Projekt ↑), das zur Lösung spezieller Aufgaben benötigt wird. Die Sicht abstrahiert, indem Informationen verkürzt werden, die zur Lösung der Aufgabe unnötig sind oder indem man Struktur und Erscheinung des Objektes konsistent so ändert, daß der Zweck der Ableitung des Partialmodelles erfüllt wird.

Simplexe

Simplexe sind Instanzobjekte, die über (noch) keine Aufbaustruktur (siehe Aggregation ↑) verfügen, d.h. sie sind zu einem bestimmten Zeitpunkt nicht durch Teilobjekte untersetzt. Sie können sehr wohl Teil von etwas sein. Sie stellen die Blätter der Aufbaustruktur zu einem bestimmten Zeitpunkt dar. Atome (↑) sind immer Simplexe. Ein Simplex ist also eine Entwicklungsstufe eines Entwurfsobjektes, alle Objekte der Aggregatstruktur sind initial Simplexe.

Slot

Wörtlich übersetzt bedeutet dies Schlitz oder Einschub. Dieser ‘Einschub’ kann ein einzelnes, beschreibendes Merkmal eines Objektes aufnehmen. Slots strukturieren das ‘Innere’ eines Objektes. Repräsentant des Slots ist sein Name. Slots gleichen Namens können jedoch bei verschiedenen Objekten verschiedene Beschreibungen tragen (Polymorphismus). Slot können Attribute, Relationen und *Methoden* aufnehmen. Die Terminologie entspricht dann der, die von der Wissensrepräsentation ‘Frame’ bekannt ist.

Strategie (Entwurfs ~)

Die Entwurfsstrategie beschreibt die Art und Weise der Lösung komplexer Entwurfsaufgaben. Bei der ‘bottom-up’-Strategie werden zuerst Detailprobleme gelöst, d.h. es werden Modellobjekte erzeugt, die diese Teilprobleme lösen. Dann werden diese Objekte schrittweise zu komplexeren Objekten aggregiert (↑), die die Geamtlösung bilden. Die ‘top-down’-Strategie ist ein Vorgehen, bei dem abstrakte Objekte rekursiv durch präzisere Teilobjekte untersetzt werden. In der Praxis der CAD-Nutzung sollten beide Strategien wechselweise nutzbar sein. Dies wird als ‘ opportunistische’ Entwurfsstrategie bezeichnet.

Super- bzw. Subklasse

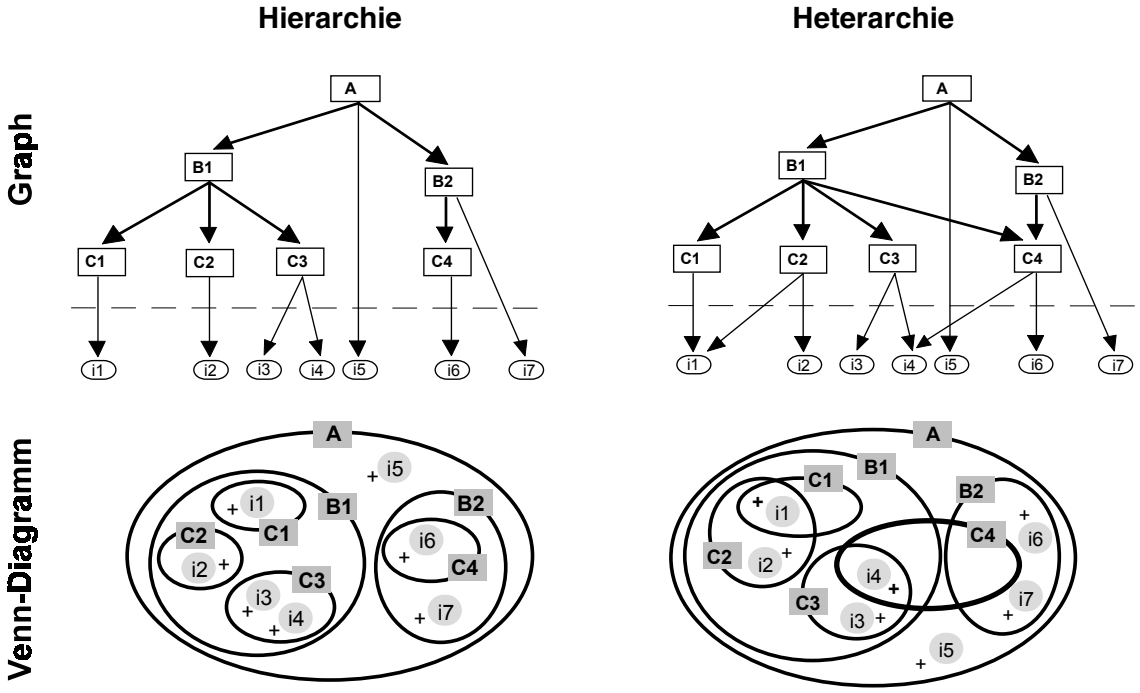
Da Klassen (↑) als Mengen möglicher Ausprägungen beschrieben werden können, kann man zwischen Klassen Mengenrelationen definieren. Eine Superklasse umfaßt neben weiteren auch alle potentiellen oder realen Ausprägungen ihrer Subklassen. Dieses System von Super-/Subklassen kann in Form eines hierarchischen bzw. heterarchischen Graphen dargestellt werden. Wenn es sich um begiffliches Wissen einer Anwendungsdomäne handelt, wird dieser Graph als Taxonomie (↓) bezeichnet. Synonyme Begriffe sind Ober- und Unterklasse.

Task

Im Rahmen des Entwurfsprozesses bezeichnet eine Task ein zu bearbeitendes Komplexobjekt (↑) im Sinne einer zu lösenden Teilaufgabe des Entwurfs. Der Taskname ist der Name eines Objektes. Alle Objekte, die einer (oder mehrerer) Tasks angehören, stellen den in Bearbeitung befindlichen Ausschnitt aus dem Projekt (↑) dar. Taskbildung dient der Reduktion der Problemkomplexität durch schlichtes Einschränken der betrachteten Objektmenge. Die Objekte selbst bleiben bei der Taskbildung unverändert.

Taxonomien

Taxonomien sind ein hierarchisches oder auch heterarchisches gegliedertes System von Klassen oder Typen. In ihnen werden Eigenschaften und Verhalten von Gruppen von Objekten verallgemeinert. Sie lassen sich als gerichteter azyklischer Graph (GAG) darstellen, dessen Kanten durch die paradigmatische Relation ‘Generalisieren/ Spezialisieren’ gebildet werden, d.h. es gibt in ihnen noch keine Instanzen. Man kann Taxonomien als eine Form der Wissensrepräsentation verstehen, sie werden häufig zur strukturierten Verwahrung begrifflichen Wissens verwendet.



Klassensysteme als Mengen

Topkomplex

siehe Komplexe ↑

Transaktion

Im Rahmen rechnergestützter Modellierung wird zwischen transienten und persistenten Modellen unterschieden. Die Überführung einer flüchtigen in eine dauerhaften Speicherform erfolgt i. allg. asynchron. Die Prozesse, die dies leisten, sind Transaktionen. Solche Transaktionen können auch einer Ausschnittsbildung komplexer in weniger komplexer Modellauschnitte oder der Bildung von Sichten dienen.

Typ

Der Begriff des Typs wird häufig im Kontext von Programmiersprachen benutzt. Mit ihm ist dort ein Konzept verbunden, wie aus einer Menge atomare Basistypen komplexere Typen zusammengesetzt werden. Für die Elemente dieses Daten-Typs erfolgt eine implizite Zuweisung von spezifischer Operatoren in Algorithmen. Typ und Klasse (↑) sind im Rahmen der Programmierung im Prinzip synonyme Begriffe.

Vererbung

Durch die Art der Mengeninklusion zwischen Super- und Subklassen (↑) erlangen alle definierten Eigenschaften der Superklasse auch Gültigkeit für die Subklasse. Dies wird als Vererbung bezeichnet und läßt sich analog auf das Verhältnis von Klasse zu Instanz (↑) ausdehnen. Vererbt werden Merkmale und/ oder Merkmalsausprägung. Bei der Vererbung von Merkmalsausprägung kann diese in den Subklassen differenziert werden, soweit sie (z.B. in der Modellbildung) nicht Grundlage der Differenzierung der Klassen selbst war.
Im Kontext der Programmiersprachen gilt die Vererbung auch für Verhalten (Methoden).

Verhalten

siehe Methode ↑

Wertfacette

Eine Wertfacette ist eine spezielle Art von Facette (↑). Falls beschreibende Merkmale (Slots↑) von Objekten durch Facetten untersetzt sind, verfügen sie i.allg. über eine Wertfacette. Sie enthält die Merkmalsausprägung, die das Merkmal tragen würde, wenn kein Facettenkonzept zugrunde läge. So wird eine homogene Verwendbarkeit facettisierter Slotobjekte im Kontext von standardisierten OO-Programmiersprachen gewährleistet, auch wenn diese nicht über ein Facettenkonzept verfügen.

Anhang B - Test zur Begriffssystematisierung

Autor: Dipl.-Ing. F. Steinmann
Datum: 12.11.1995

Um zu prüfen, ob für potentielle Nutzer von CAAD-Systemen Taxonomien tatsächlich eine 'natürliche' Strukturierung ihres begrifflichen Wissens darstellt, wurde ein Test entworfen, bei dem durch Probanden 98 willkürliche gewählte Begriffe des Bauwesen geordnet werden sollten. Die zu schaffende Ordnung sollte die inhaltlichen Beziehungen der Begriffe gut wiedergeben.

Es nahmen 9 Personen teil (ein Student, 3 Bauingenieure, 5 Architekten). Lediglich 2 Bauingenieure (B1, B2) und 2 Architekten (A1, A2) beendeten jedoch den Test.

Es folgen die Aufgabenstellung, sowie einige ausgewählte Ergebnisse in Form skizzierter Taxonomien. Eine Auswertung der Ergebnisse erfolgt im Kapitel 6 der Dissertation.

Test zur Begriffssystematisierung

Sie finden nachfolgend eine Reihe von Begriffen des Bauwesens. Die Begriffe sind mehr oder weniger zufällig ausgewählt und liegen ungeordnet vor. Trotzdem werden Sie zwischen ihnen inhaltliche Ähnlichkeiten, Beziehungen und Abhängigkeiten entdecken. Schaffen Sie sich ein System zum Ordnen dieser Begriffe, das ihrer Meinung nach die Beziehungen der Begriffe untereinander gut repräsentiert. Ordnen Sie die Begriffe in dieses System ein und ergänzen Sie bei Bedarf fehlende Begriffe. Kennzeichen Sie bitte die von Ihnen hinzugefügten Begriffe.

Danke

BEGRIFFSLISTE :

Arbeitszimmer
Außentür
Autobahn
Bad
Badewanne
Balken
Bauwerk
Besprechungszimmer
Bett
Brücke
Büro
Bürohaus
Dach
Dachbinder
Dachgeschoß
Dachpfette
Dachsparren
Decke
Diele
Doppelhaus
Dusche
Duschkabine
Eifelturm
Einfamilienhaus
EmpireState_Building
Erdgeschoß
Eßzimmer
Fachwerkbrücke
Fahrstuhl
Fenster
Fitnessraum
Flur
Fundament
Fußweg
Garage
Garten
Garten
Gästezimmer
Hängebrücke
Haus_am_Horn
Heizung
Herd
Hobbyraum
Hochbauten
Holzbrücke
Keller
Kinderzimmer
Konferenzraum
Küche
Kundentoilette
Lagerbereich
Lagerraum
Landstraße
Möbel
Normalgeschoß
Personaltoilette
Podest
Rampe
Reihenhaus
Rezeption
Ringanker
Ruheraum
Sauna
Schlafzimmer
Schornstein
Schrank
Schwitzraum
Seil
Sekretariat
Sessel
Sitzgelegenheit
Sofa
Solarium
Spüle
St.Gotthardt_Tunnel
Straße
Stufe
Stuhl
Stütze
Swimmingpool
Tauchbecken
Terrasse
Teufelstalbrücke
Tiefbauten
Tisch
Toilette
Towerbridge
Treppen
Treppenhaus
Tunnel
Turm
VanDeVelde_Bau
Veranda
Verkaufsbereich
Wand
Wasserbehälter
Werkstatt
Wohnbereich
Zufahrt

EXEMPLARISCHE AUSSCHNITTE AUS DEN ERGEBNISSEN:

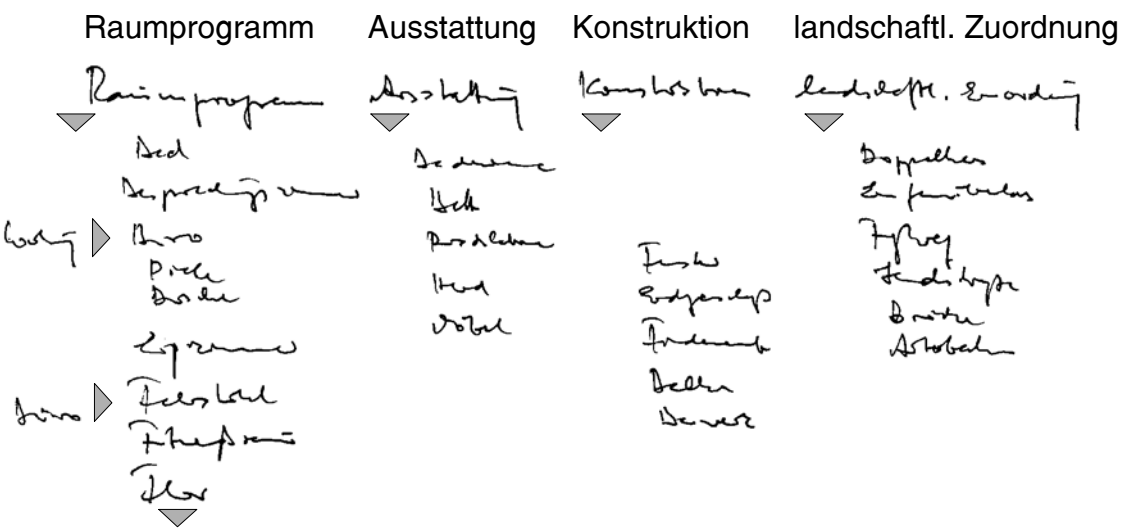


Abb. B1 Ausschnitt aus der Lösung von B. KOLBE (Architekt)

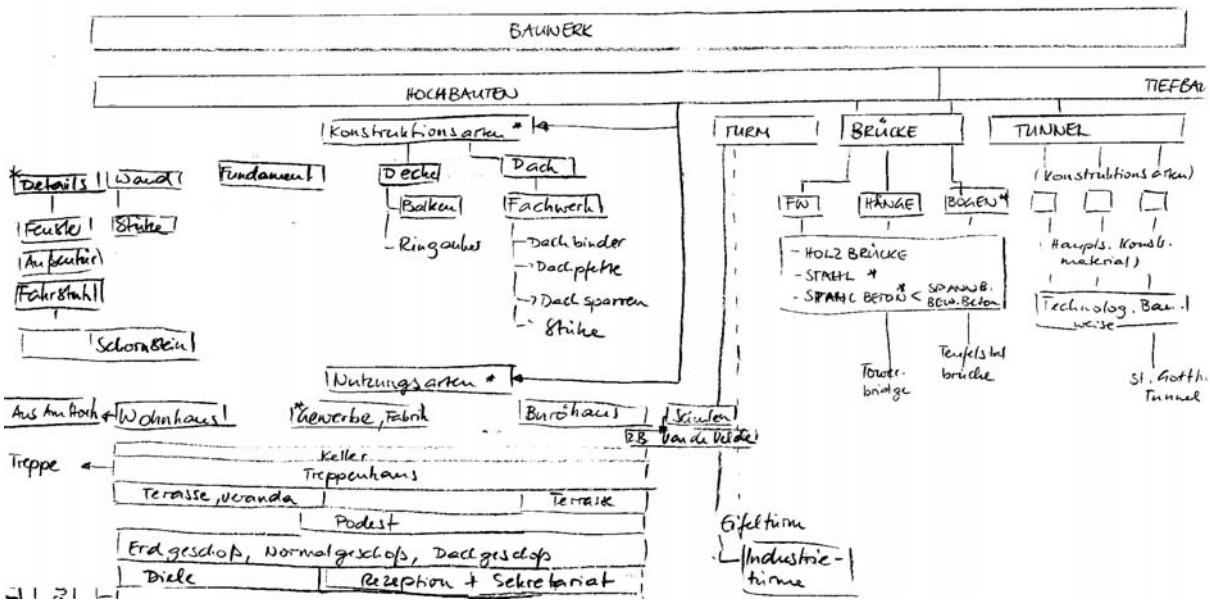


Abb. B2 Ausschnitt aus der Lösung von H. KLENNER (Bauingenieur)

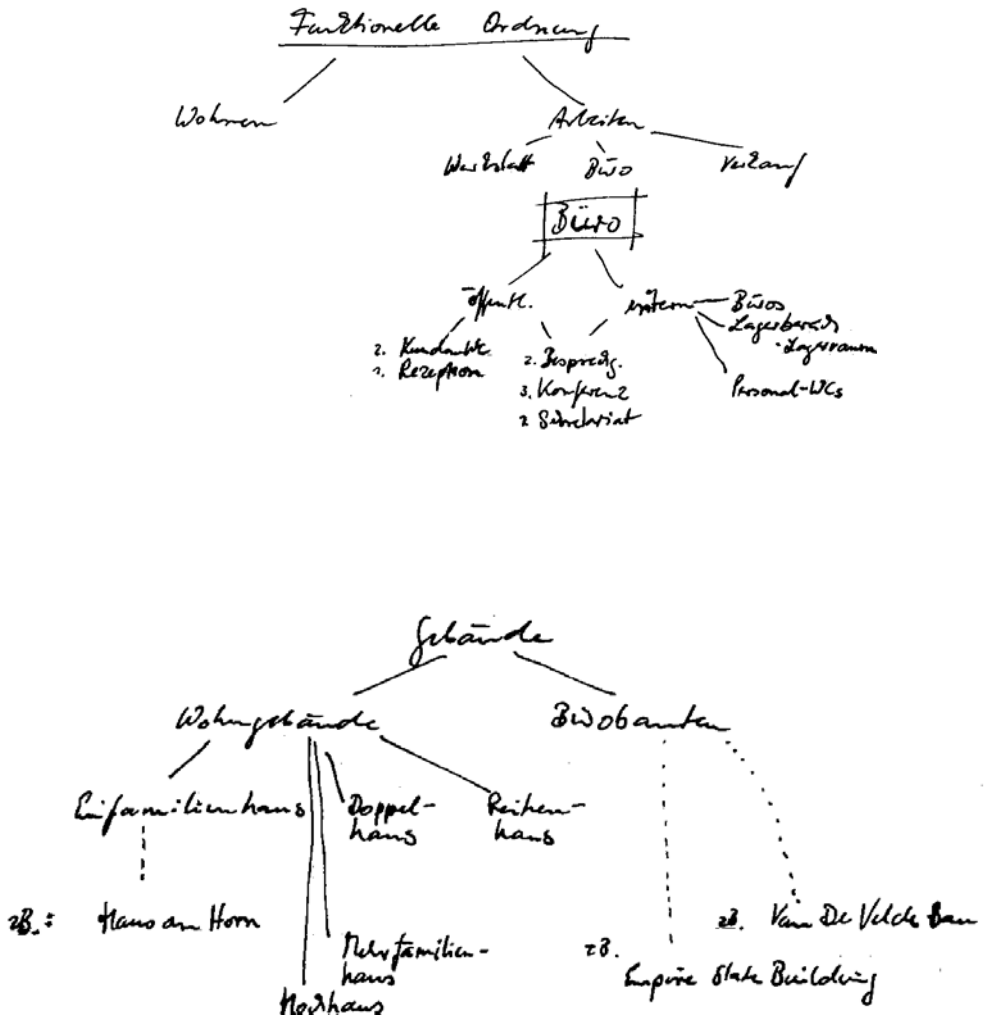


Abb. B3 a,b Ausschnitte aus der Lösung von Dr. I. LÜTZKENDORF (Architekt)

Anhang C - Dokumentation FUNPLAN (Version 2.1, März '95)

C1 - Installation

Autoren: Dipl.-Inf. P. Kolbe
Dipl.-Ing. F. Steinmann

Hardwarevoraussetzungen: IBM-kompatibler PC/AT, Prozessor 80486/66 Mhz, Hauptspeicher 16 MB, SVGA

Softwarevoraussetzungen : MS-DOS **6.2 /** MS-Windows 3.1 , Windows NT, SMARTDRV.EXE (bei DOS), AutoCAD 12 oder 13 für DOS oder Windows

Zur Ausführung notwendige Dateien:

EGAVGA.BGI	FRAGE.BMP	FPDESIGN.LSP	FPSYM.LSP
SANS.CHR	TITEL.BMP	FPFILTER.LSP	FPDDE.LSP
TRIP.CHR	INFO.TXT	FPLAYER.LSP	FPEDITRE.LSP
ZGO.CFG	FUNPRO.SLB	FPTAXO.LSP	FPUNDO.LSP
ZGO.EXE	FPROJEKT.DCL	FPZSHOW.LSP	FPGLOBAL.LSP
	FUNPLAN.KAL	FPACAD.LSP	FPNUM.LSP
	ACAD.DWG	FPEDITBE.LSP	FUNPRO.LSP
	ACAD.LSP	FPAGG.LSP	FPINST.LSP
HILFE.EXE	ACAD.MNX	FPKUM.LSP	FPSTATE.LSP
HELPMAN.EXE	FUNPLAN.MNX	FPFIDLO.LSP	FPWISS.LSP
FUNPLAN.HLP	FPREPORT.LSP	FPZREL.LSP	FPRAUMB.LSP

Das gesamte AutoCAD Laufzeitsystem muß verfügbar sein.
Für KappaPC genügt zur Ausführung die KAPPARUN.EXE (KappaPC - Laufzeitumgebung)
Die angegebenen Dateien sollten in ein eigenes Verzeichnis kopiert werden
(im folgenden Beispiel ist dies D:\USR \AKTUELL).

Folgende AutoLISP-Erweiterungen **ALOS** und **ALAS** müssen verfügbar sein :

ALOS.LSP	AOPRED.LSP	ALAS.LSP
AOINIT.LSP	AOMONITO.LSP	AAANALY.LSP
AOANALY2.LSP	AOLOESCH.LSP	AAEDIT.LSP
AOERROR.LSP	AOHILF.LSP	AAPRED..LSP
AOANALY1.LSP	AOERZEUG.LSP	AAZEIG.LSP
AOZEIG.LSP	AOSENDE.LSP	AABASIS.LSP
AOCOPY.LSP	AOITAB.LSP	AALOOP.LSP
AOAENDER.LSP	AOLASI.LSP	AAORG.LSP
AOZUGRI.LSP	AOREVIEW.LSP	

Diese Dateien müssen in das AutoCAD-Verzeichnis ... \ACADWIN\SUPPORTkopiert werden.

- Erzeugen der Windows-Ikonen:
  Programm-Manager: Datei/Neu ... /Programmgruppe wählen
  Beschreibung: "FunPlan"
  OK-Schalter zum Bestätigen

  Programm-Manager: Datei/Neu ... /Programm wählen
  Beschreibung:"KAPPA-PC - FunPlan/Wissen"
  Befehlszeile:
  "E:\ KAPPA22\KAPPARUN.EXE D:\USR\AKTUELL\FUNPLAN.KAL“
  wobei der erste Dateiname das ausführbare KAPPA-PC
  und der zweite das KAL-Quelltextfile ist.
  Arbeitsverzeichnis: "D:\USR\PKO\AKTUELL'
  Ikone wählen, OK-Schalter zum Bestätigen
  Programm-Manager: Datei/Neu ... /Programm wählen
  Beschreibung:"AutoCAD - FunPlan/Projekt"
  Befehlszeile: „E:\ACADWIN\ACAD.EXE FUNPLAN"
  Arbeitsverzeichnis: "D:\USR\AKTUELL"
  Ikone wählen, OK-Schalter zum Bestätigen

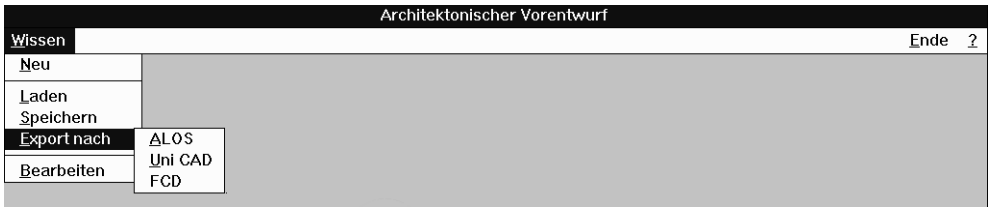
- In der Datei FUNPRO.LSP sind die Pfade für AutoCAD und KAPPA-PC anzupassen.

Anhang C - Dokumentation FUNPLAN (Version 2.2, Okt.'95)

C2 - Wissensmodul (Kurzreferenz)

Autor: Dipl.-Ing F. Steinmann

Menü Wissen



Neu Anlegen einer neuen, leeren Wissensbasis. Der Wissensbasisname wird als Name der Wurzelklasse benutzt.

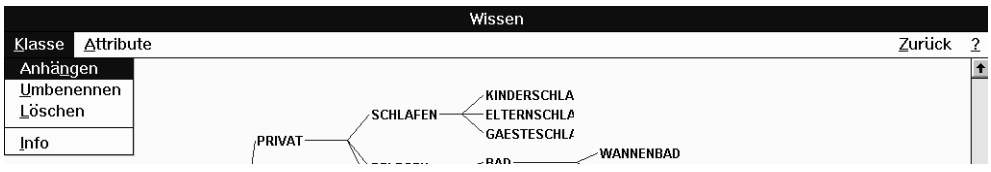
Laden Es erscheint eine Dateiauswahlbox zur Selektion einer gespeicherten Wissensbasis (Erweiterung '*.AVE')

Speicher Es erscheint eine Dateiauswahlbox zur Selektion einer gespeicherten Wissensbasis (Erweiterung '*.AVE') bzw. zum Eingeben eines neuen Dateinamens für die zu speichernde Wissensbasis.

Export nach Es erscheint eine Dateiauswahlbox zum 'Speichere'-Dialog. Um Wissensbasen mit FUNPLAN-Projekt zu nutzen, müssen diese nach FCD exportiert werden ('file of classdefinition'). Der Export nach ALOS bzw. UniCAD ist gegenwärtig gesperrt.

Bearbeiten Die im Speicher befindliche Wissensbasis wird bearbeitet. Der Klassenbrowser wird angezeigt.

Menü Klassen *- (redundant zum 'Click-menü' Klassen)*



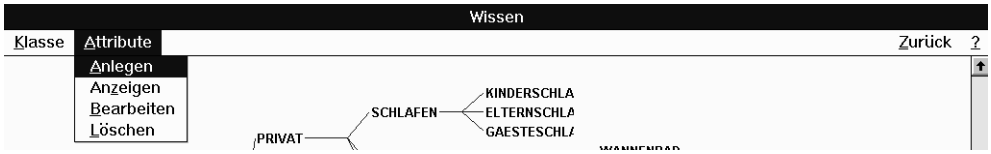
Anhängen An die im Klassenbrowser selektierte Klasse wird eine neue Subklasse angehangen. Es dürfen keine Klassennamen mehrfach verwendet werden.

Umbenennen Die im Klassenbrowser selektierte Klasse wird umbenannt.

Löschen Die im Klassenbrowser selektierte Klasse wird gelöscht. Die Klasse darf keine Subklassen besitzen. Wenn doch, sind erst alle ihre Subklassen zu löschen.

Info Unter diesem Menüpunkt wird die jeweilige Implementation von INFPLAN aufgerufen. Sie zeigt nichtformalisierte Daten wie Text, Klänge, oder eben beliebige Dokumente für die jeweils selektierte Klasse an. Das Programm ist nicht immer verfügbar.

Menü Attribute *- (redundant zum 'Click-menü' Attribute)*



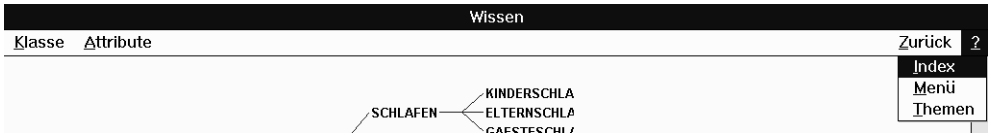
Anlegen Bei der im Klassenbrowser selektierten Klasse wird ein neues Attribut oder eine neue Relation angelegt. Es folgt ein Submenü zur Auswahl der Attribut- bzw. Relationentypklasse.

Anzeigen Bei der im Klassenbrowser selektierten Klasse werden alle verfügbaren Attribute und Relationen angezeigt (auch ererbte).

Bearbeiten Bei der im Klassenbrowser selektierten Klasse kann unter allen verfügbaren Attributen und Relationen (auch ererbte) eine ausgewählt werden und dann deren Belegung für alle Wert-Facetten (z.B. Minimum, Maximum usw.) bearbeitet werden. Hiermit können ererbte Werte überschrieben werden. Neue Werte werden auf Konsistenz zu ihrer Superklasse getestet.

Löschen Bei der im Klassenbrowser selektierten Klasse kann unter allen dort definierten Attributen und Relationen (nicht der ererbten) eine ausgewählt werden und diese gelöscht werden. Sie ist dann auch bei allen Subklassen nicht mehr verfügfbar!

Menü ?



?

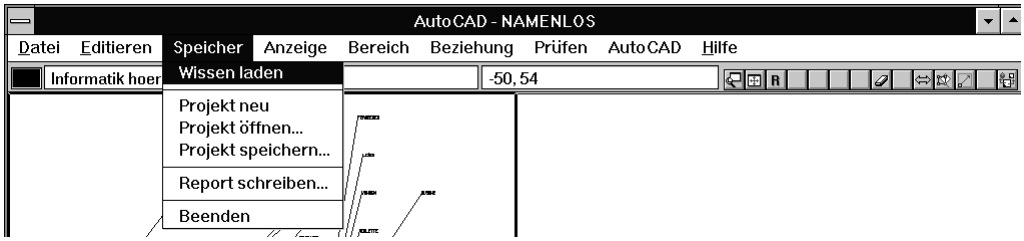
(Hilfe) Die Hilfe zu FUNPLAN-Wissen mit der Windows-üblichen Funktionalität wird aufgerufen

Anhang C - Dokumentation FUNPLAN (Version 2.3, Jan. '96)

C3 Projektmodul (Kurzreferenz)

Autoren: Dipl.Inf P. Kolbe
Dipl.Ing F. Steinmann

Menü Speicher



Wissen laden Durch den Befehl "Wissen laden..." kann eine neue Wissensbasis geladen werden. Wissensbasen werden durch FunPlan/Wissen erstellt und geändert. Die Dateierweiterung lautet *.FCD.
Achtung: Alle Projektdaten werden gelöscht.

Projekt neu Soll das momentan bearbeitete Projekt verworfen und ein neues begonnen werden, wählen Sie den Befehl "Projekt neu". Dabei werden alle Projektdaten gelöscht.

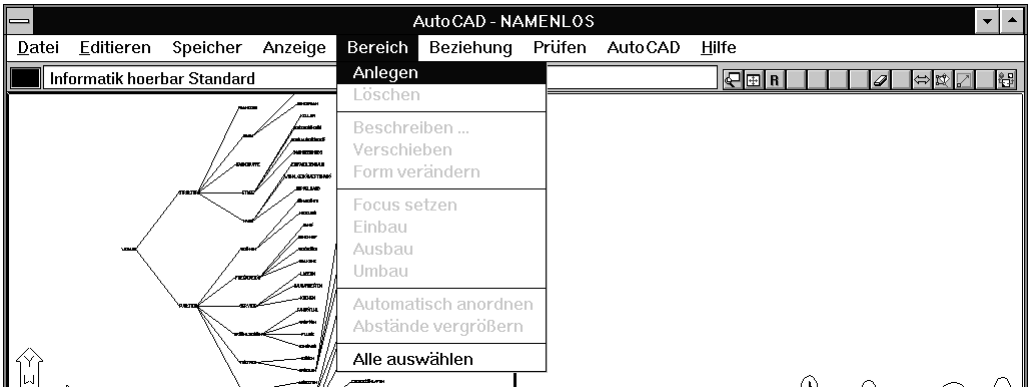
Projekt öffnen Um ein gespeichertes Projekt (FID-Format) zu laden, wählen Sie den Befehl "Projekt öffnen". Ein bestehendes Projekt kann auch zu dem aktuell bearbeiteten Projekt hinzugeladen werden. Die neuen Entwurfsteile werden in der Aufbaustruktur als Teile des Bereiches angelegt, der als Focus gekennzeichnet ist. Anschließend werden alle feinsten Teilbereiche (Simplexe) angezeigt. Beim Laden eines Projektes erfolgt die Zuordnung der Entwurfsteile zu den vorhandenen Klassen der Wissensbasis. Projektdaten können an beliebige Wissensbasen angepaßt werden. Wurde eine andere Wissensbasis geladen, als die zur Erstellung des Projektes genutzte, können einzelne Merkmale entfallen bzw. Zuordnungen zu Klassen verändert werden. Es bleiben jedoch alle Entwurfsteile (Objekte) vorhanden.

Projekt speichern Durch den Befehl "Projekt speichern" wird das momentan bearbeitete Projekt gespeichert. Dabei werden alle angelegten Bereiche, die beschriebenen Merkmale und Beziehungen abgelegt. Das Dateiformat ist '*.FID'.

Report schreiben Durch den Befehl "Report schreiben" wird das momentan bearbeitete Projekt in ein strukturiertes ASCII-File gespeichert. Dabei werden alle angelegten Bereiche, die beschriebenen Merkmale und Beziehungen abgespeichert. Das Dateiformat ist '*.TXT'. Die Datei kann anschließend mit einem Textprozessor weiterbearbeitet werden, z.B. als Raumbuch.

Beenden Der Befehl "Beenden" schließt die Anwendung FunPlan/Projekt. Zuvor sollten Sie die Projektdaten sichern. Während jeder Sitzung entsteht aus den Daten eine Zeichnung. Diese Zeichnung ist stets wieder herstellbar. Soll diese Zeichnung ausgeplottet werden, so ist dies mit AutoCAD-Befehlen möglich. Wählen Sie dazu vor dem Beenden den Befehl "AutoCAD" um von FunPlan/Projekt zu AutoCAD zu wechseln.
Achtung: Zeichnungsänderungen, die mit AutoCAD-Befehlen durchgeführt werden, haben keine Auswirkung auf die Daten von FunPlan/Projekt.

Menü Bereich



Anlegen Durch den Befehl "Anlegen" können Sie ein neues Entwurfsteil (einen Funktionsbereich) erzeugen. Sie werden zunächst nach einer Klasse gefragt, welche in der Klassenstruktur (linke obere Abbildung) durch Zeigen gewählt werden kann. Anschließend werden Sie zur Eingabe der Position und Ausdehnung aufgefordert. Danach öffnet sich ein Dialogfenster, worin für den neuen Bereich ein Name vergeben werden kann. Desweiteren ist es möglich, die Merkmale des Bereichs zu beschreiben. Das neu angelegte Funktionsobjekt wird als Teilbereich des Bereichs erfaßt, der momentan durch den Focus gekennzeichnet ist.

Löschen Der Befehl "Löschen" entfernt einen oder mehrere zuvor ausgewählte Funktionsbereiche. Das Löschen ist nur möglich, wenn der zu löschende Bereich nicht durch Teilbereiche untergliedert ist. Nach erfolgreichem Löschen werden alle anwenderdefinierten Beziehungen zu diesem Bereich entfernt. *Achtung*: In dieser Programmversion ist ein Wiederherstellen eines gelöschten Bereiches nicht möglich!

Beschreiben Nach Selektion eines Funktionsbereiches öffnet der Befehl "Beschreiben..." das Dialogfenster zur Beschreibung eines Funktionsbereiches. Neben allen anwenderdefinierten Merkmalen können hier der Name des Funktionsbereiches, die allgemeine Zuordnung (Klasse) und ein beschreibender Kurztext verändert werden.

Verschieben Mit dem Befehl "Verschieben" können selektierte Funktionsbereiche verschoben werden. Nach Befehlswahl werden Sie zur Eingabe des neuen Mittelpunktes aufgefordert. Mit diesem Befehl können gleichzeitig beliebig viele sichtbare Funktionsbereiche verschoben werden.

Form verändern Durch den Befehl "Form verändern" kann die Form eines zuvor ausgewählten Funktionsbereiches verändert werden. Der Mittelpunkt bleibt gleich, wogegen das Verhältnis von Länge zu Breite geändert wird. Diese Eingabe wirkt sich direkt auf das nutzerdefinierte Attribut "flaeche" für die Randbedingung "Standard" aus.

Focus setzen Der Begriff Focus wird verwendet, um den Bereich zu kennzeichnen, der gerade verfeinert wird. Bei einem neuen Projekt wird der Focus auf das zuerst angelegte Entwurfsteil gesetzt. Alle folgenden neuen Entwurfsteile werden als Teilbereiche dieses ersten Bereiches angesehen. Um solch ein folgendes Teil weiter zu untergliedern, muß der Focus darauf gesetzt werden. Der Focus hat auf folgende Befehle Einfluß: "Bereich Anlegen", "Einbau", "Ausbau", "Umbau" und "Projekt öffnen". Um den Focus auf einen anderen Bereich zu setzen, muß zunächst ein Bereich in der Anzeige der Aggregationsstruktur gewählt werden und anschließend der Befehl "Focus setzen". In der Aggregationsstruktur wird der Focus blau gezeichnet. Außerdem ist in der Titelzeile der Name des Bereichs angegeben, worauf gerader der Focus zeigt.

Einbau Durch den Befehl Einbau kann ein Bereich zusätzlich einem anderen Bereich als Teilbereich zugeordnet werden. Anschließend verfügen zwei Bereiche über denselben Teilbereich. Setzen Sie zunächst den Focus auf den Bereich, in den etwas eingebaut werden soll. Wählen Sie anschließend in der Aggregationsstruktur den Bereich und den Befehl "Einbau". Daraufhin wird die Aggregationsstruktur neu gezeichnet.

Ausbau Um einen mehrfach eingebauten Bereich wieder auszubauen, ist zunächst der Focus auf den Bereich zu setzen, aus dem etwas ausgebaut werden soll. Selektieren Sie dann in der Aggregationsstruktur den Bereich und den Befehl "Ausbau".

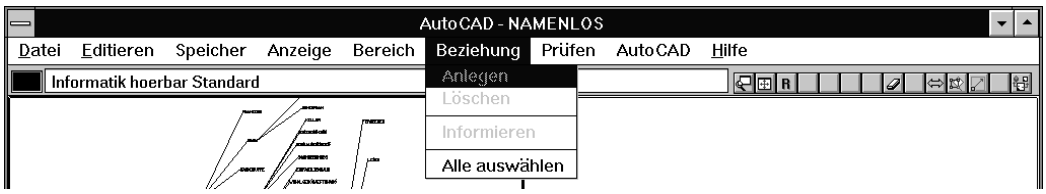
Umbau Durch den Befehl "Umbau" ist es möglich einen Bereich auszubauen und in einen anderen Bereich wieder einzubauen. Setzen Sie vorher den Focus auf den Bereich, in den etwas eingebaut werden soll, selektieren Sie den umzubauenden Bereich in der Aggregationsstruktur und wählen Sie anschließend den Befehl "Umbau". War der Bereich in mehrere Bereiche ausgebaut, so wird er aus allen ausgebaut und nur dem Focus-Bereich als Teil zugeordnet.

Automatisch anordnen Durch den Befehl "Automatisch anordnen..." kann die Position der Funktionsbereiche optimiert werden. Es wird versucht, entsprechend der gewählten Beziehungen, möglichst kurze Verbindungen und wenige Kreuzungen zu erzeugen. Das Ergebnis kann jeweils verschieden ausfallen. Selektieren Sie zunächst Beziehungen zwischen den zu verändernden Bereichen. Nach Befehlswahl öffnet sich ein Dialogfenster, in dem Sie eine grobe Form angeben können, in der alle Bereiche angeordnet werden. Nach Drücken des OK-Schalters beginnt ein Programm die Optimierung - es erfolgen Grafikausgaben. Wenn in der linken oberen Ecke "100%" angezeigt wird, ist die Optimierung beendet - drücken Sie dann eine Taste, um die Bearbeitung fortzusetzen.

Abstände vergrößern Um zwischen mehreren Bereichen etwas Platz zu schaffen, wählen Sie diese Bereiche und anschließend den Befehl "Abstände vergrößern". Die Bereiche werden dann proportional auseinandergerückt.

Alle auswählen Verwenden Sie "Alle auswählen", um alle sichtbaren Funktionsbereiche zu selektieren.

Menü Beziehung



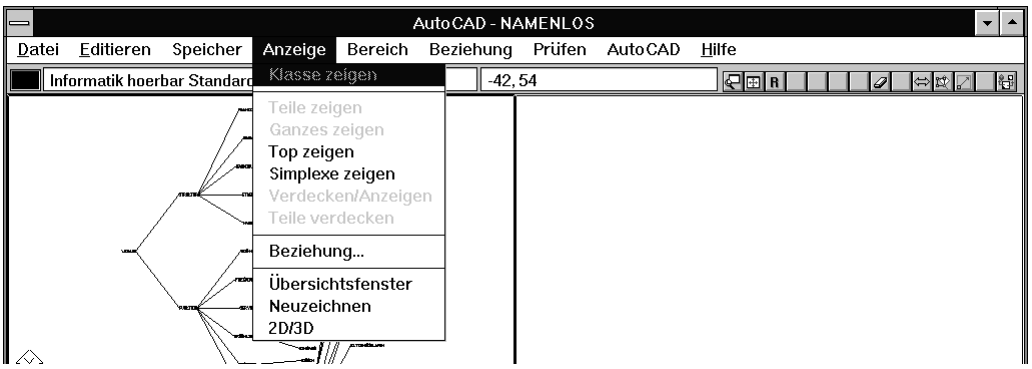
Anlegen In einem Funktionsplan können neben den Funktionsbereichen Beziehungen zwischen den Bereichen erfaßt werden. Unter dem Befehl "Anzeige/Beziehung.." können die aktuell zu bearbeitende Beziehung und Randbedingung eingestellt werden - diese Angaben sind stets in der Titelzeile sichtbar. Um eine Beziehung anzulegen, wählen Sie zunächst die beiden Bereiche, zwischen denen die Beziehung erfaßt werden soll, und anschließend den Befehl.

Löschen Wählen Sie durch Anklicken die Funktionsbereiche bzw. Beziehungen, um sie mit dem Befehl "Löschen" zu entfernen.

Informieren Der Befehl "Informieren" zeigt für eine Beziehung die betreffenden Funktionsbereiche, die Art der Beziehung und die Randbedingung an. Dieser Befehl ist wählbar, wenn eine oder mehrere Beziehungen selektiert wurden. Allerdings wird jeder Beziehungstyp nur einmal angezeigt, auch wenn mehrere gewählt wurden.

Alle auswählen Verwenden Sie "Alle auswählen" um alle sichtbaren sichtbaren nutzerdefinierten Beziehungen (Assoziationen) zu selektieren.

Menü Anzeige



Klasse zeigen Nach der Auswahl einer Klasse in der Klassenstruktur (linke obere Abbildung) können die Merkmale und ihre Belegungen einer Klasse angezeigt werden. Über diese Werte verfügen die Instanzen nach dem Anlegen als Voreinstellung.

Teile zeigen Ist ein Bereich durch Bereiche untergliedert, so können durch den Befehl "Teile zeigen" die Teile sichtbar und der Bereich selbst unsichtbar gemacht werden. Ein sichtbarer Bereich wird in der Anzeige der Aggregationsstruktur umrahmt und ist in der Anzeige der Funktionsbereiche und Assoziationen (nutzerdefinierte Beziehungen) sichtbar - ansonsten wird nur der Name in der Aggregationsstruktur angezeigt. Vor der Befehlsauswahl muß der gewünschte Bereich in der Aggregationsstruktur selektiert werden. Der entgegengesetzte Befehl lautet "Ganzes zeigen".

Ganzes zeigen Um für einen Teilbereich den Bereich zu zeigen, in dem er eingebaut ist, muß in der Aggregationsstruktur dieser Bereich gewählt werden und anschließend der Befehl "Ganzes zeigen". Dabei wird der übergeordnete Bereich angezeigt und alle Teile dieses Bereichs ausgeblendet. Der entgegengesetzte Befehl lautet "Teile zeigen".

Top zeigen Durch den Befehl "Top zeigen" wird der zuerst angelegte Funktionsbereich gezeigt - alle anderen Bereiche werden unsichtbar.

Simplexe zeigen Als Simplexe werden alle Bereiche bezeichnet, die in der Aufbaustruktur durch keine weiteren Bereiche untergliedert sind. Der Befehl "Simplexe zeigen" zeigt diese Bereich - alle anderen Bereiche werden unsichtbar.

Verdecken/ Anzeigen Um die Sichtbarkeit (im Funktionsplangraphen) eines Bereiches zu ändern, wählen Sie in der Aggregationsstruktur diesen Bereich und den Befehl "Verdecken/Anzeigen". Daraufhin wird dieses Entwurfsteil sichtbar, wenn es unsichtbar bzw. unsichtbar, wenn es sichtbar war.

Teile verdecken Der Befehl "Teile verdecken" macht alle Teilbereiche des gewählten Bereichs unsichtbar.

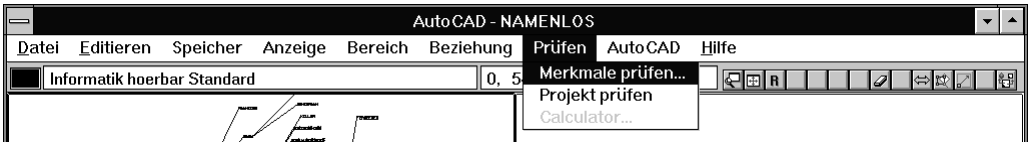
Beziehungen ... Durch den Befehl "Beziehung..." läßt sich die Anzeige der Assoziationen steuern. Es besteht die Möglichkeit, eine Beziehnung als die aktuelle Beziehung auszuwählen - alle nachfolgend angelegten Beziehungen sind von diesem Typ. Desweiteren läßt sich eine Randbedingung wählen - anschließend sind nur die Beziehungen dieser Randbedingung sichtbar und die nachfolgend angelegten Beziehungen gelten für diese Randbedingung. Es ist möglich, die Menge der sichtbaren Beziehungen einzuschränken und für jede Beziehung kann eine Farbe definiert werden. Achtung: Bei Programmende gehen diese Einstellungen verloren.

Übersichtsfenster Durch den Befehl "Übersichtsfenster" wird unter AutoCAD für Windows ein Fenster geöffnet, worin ZOOM-Befehle für die dargestellten Strukturen möglich sind. Unter AutoCAD für DOS wird an dieser Stelle der ursprüngliche AutoCAD-Zoom-Befehl mit der Option ‘dynamisch’ aufgerufen. Die Eingabe im Übersichtsfenster kann jederzeit - auch während der Aufforderung zur Auswahl einer Klasse - genutzt werden.

Neuzeichnen Nutzen Sie den Befehl "Neuzeichnen", um Ihre grafische Darstellung zu regenerieren. Diese Aktion wird nicht automatisch ausgeführt, da ansonsten der Bildschirm ständig "flackern" würde. Diesen Befehl können Sie auch durch das Kürzel "N" über die Tastatur eingeben.

2D/3D Durch den Befehl "2D/3D" wird zwischen der Draufsicht (2D) und einer beliebigen 3D-Ansicht für die Darstellung des Funktionsplans gewechselt.

Menü Prüfen

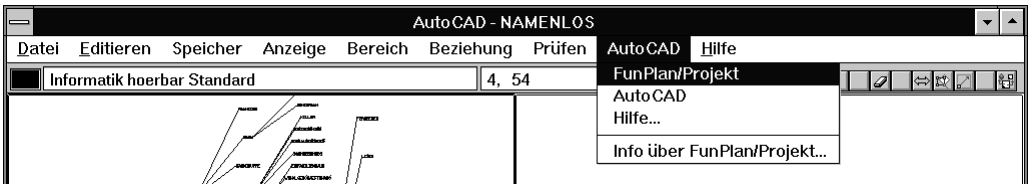


Merkmale prüfen Durch den Befehl "Merkmale prüfen..." läßt sich die Menge der zu prüfenden Attribute einschränken.

Projekt prüfen Durch den Befehl "Projekt prüfen..." werden alle aktiven und nicht aktiven Randbedingungen für Relationen und Attribute des Projektes geprüft (kann einige Zeit dauern).

Calculator Mit dem Befehl "Calculator..." wird ein Dialogfenster geöffnet, worin die Auswahl von einem oder zwei anwenderdefinierten numerischen Attributen möglich ist. So können beispielsweise die Flächen für alle Teilbereiche eines gewählten Bereichs oder die Fläche multipliziert mit dem Quadratmeterpreis ermittelt werden.

Menü AutoCAD



FunPlan/Projekt Sollte das Programm FunPlan/Projekt "abgestürtzt" sein - zu erkennen an der fehlenden Aufforderung "FunPlan/Projekt: Bitte Objekt oder Menübefehl wählen!" in der untersten Bildschirmzeile - dann können Sie durch den Befehl "FunPlan/Projekt" das Programm wieder starten. In den meisten Fällen gehen dabei Ihre Daten **n i c h t** verloren.

AutoCAD Durch den Befehl "AutoCAD" geht das Programm FunPlan/Projekt zu AutoCAD über. Sie erhalten so die vollständige Befehlsmenge von AutoCAD. Dann können Sie die erhaltene Zeichnung mit beliebigen Befehlen verändern und ausplotten. Beachten Sie jedoch, daß die Änderungen an der AutoCAD-Zeichnung keine Auswirkungen auf Ihr Datenmodell in FunPlan/Projekt besitzt. AutoCAD bearbeitet Zeichnungen im DWG-Dateiformat. Mit FunPlan/Projekt werden Gebäudedaten im FID-Format erarbeitet. Quasi als "Nebenprodukt" entsteht die AutoCAD-Zeichnung im DWG-Format. Wenn Ihr Projekt durch FunPlan gespeichert wurde, können Sie es wieder laden und es entsteht die AutoCAD-Zeichnung - deshalb brauchen Sie eigentlich keine Zeichnung im Format DWG zu speichern. Durch den Befehl "FunPro" gelangen Sie von der AutoCAD-Befehlszeile wieder zu FunPlan/Projekt.

Hilfe ... Anzeige der FunPlan - Hilfe (im AutoCAD-Hilfeformat)

Info über FunPlan/Projekt Willkommens-Gruß und Copyright-Vermerk

Anhang C - Dokumentation FUNPLAN (Version 2.1, März '95)

C4 - Schnittstellen

Autoren: Dipl.-Inf. P. Kolbe
Dipl.-Ing. F. Steinmann

Semantik und Pragmatik

Mit dieser Sprache wird eine programmiersprachunabhängige Darstellung von Klassensystemen und Instanzbeschreibungen angestrebt, vor allem für eine permanente Speicherung von Modelldaten (Instanze) und Domänwissen (Klassen). Durch die Klassen wird allgemeingültiges Wissen beschrieben; in den Instanzenbeschreibungen sind Projektinformationen enthalten. Da mit einer Wissensbasis mehrere Projekte erstellt werden können, ist die Trennung von Wissensbeschreibung und Projektdaten sinnvoll. Projektdaten können mit beliebigen Wissensbasen bearbeitet werden, allerdings kann es dabei zu Informationsverlusten bei den Projekten kommen. Der Vorteil entsteht bei Änderung der Wissensbasis und Weiterbearbeitung der Projektdaten.

Für den Aufbau der Dateien gilt, daß die Klassenbeschreibung nach dem Kopf, bestehend aus Topic, Datum und eventuellem Kommentar eine Beschreibung beliebig vieler Klassen enthält. Die Instanzbeschreibung enthält nach Topic, Datum und eventuellem Kommentar die Beschreibung der Instanzen. In den Klassen können Attribute definiert werden und mit Anfangsbelegungen (Attribut setzen) versehen werden. Bei den Instanzen können keine weiteren Attribute definiert werden, es ist nur das Setzen von Attributen erlaubt.

Die erste definierte Klasse entspricht der Wurzelklasse. Alle folgenden Klassen müssen als direkte oder indirekte Unterklassen definiert sein. Für die Reihenfolge der Klassendefinition gilt, daß die Oberklasse stets vor der Unterklasse definiert wird. Instanzen beschreiben ihre Zugehörigkeit zu einer Klasse als Liste von Klassennamen, wobei die erste die eigentliche Klasse ist. Alle weiteren sind jeweils Oberklassen, bis zur letzten Klasse, der Wurzelklasse der Taxonomie. Somit kann eine Instanz auch bei Fehlen der Klasse der nächsten entsprechenden zugeordnet werden. Ist gar keine der angegebenen Klassen mehr vorhanden, wird die Instanz der neuen Wurzelklasse zugeordnet. Somit entsteht kein Verlust von Instanzen eines Projektes. Allerdings kann die Instanz nach der Zuordnung zu einer anderen Klasse nur über die Attribute verfügen, die in der neuen Klasse definiert sind.

Attribute gehören einem der in Tabelle A4-1 aufgeführten Attribut- und Relationentypen an. Sie können die Cardinalität einfach, Vektor (feste Feldlänge) oder Menge (variable Anzahl) besitzen.

Es sind scharfe (CRISP) und unscharfe (FUZZY) Beschreibungen möglich. Desweiteren können Attribute durch ein Randbedingungssystem beschränkt sein (CONSTRAINT) oder auch nicht (UNCONSTRAINT). Die Beschreibung (Definition) des Attributs erfolgt nur in der Klasse, in der das Attribut erstmals definiert wird. Für numerische Werte ist die Angabe einer Maßeinheit möglich.

Die Vererbung von Attributen kann wie folgt gesteuert werden:

- **NON**: Es existiert keine Vererbung, das Attribut ist nur in dieser Klasse und ihren Instanzen verfügbar.
- **STANDARD**: Das Attribut wird allen Unterklassen vererbt.
- **EXTENTABLE**: Für das Attribut gelten die Regeln der Randbedingungen. Hiermit kann eine Spezialisierung der Wertebereiche beschrieben werden.

Bei der Definition der Attribute können Wertebereiche angegeben werden. Insbesondere für symbolische Attribute werden hier die Menge der möglichen Werte beschrieben. Die erlaubten Werte für relationale Attribute sind Teilmengen der Menge aller Klassen.

Für jede Instanz ist der Slot HAS-PART und PART-OF besetzt. Sie beschreiben den Aufbau von Komplexobjekten. Die Reihenfolge der Instanzen ist dahingehend festgelegt, daß die erste Instanz das Gesamtprojekt beschreibt und alle weiteren Instanzen direkt oder indirekt Teile dieser Instanz sind und der Verweis eines PART-OF-Slots stets auf vorangegangene Instanzen zeigt.

Syntax

- Schnittstelle als ASCII-Datei mit 7-bit-Zeichensatz
- Dateierweiterung für Klassenbeschreibung *.FCD, für Instanzen *.FID
- Schlüsselwörter stets Großbuchstaben (max. 20 Zeichen)
- Trennung der Bezeichner, Konstanten und Schlüsselwörter immer durch CRLF
- immer linksbündig, d.h. keine fahrenden Leerzeichen oder Tabulatoren
- maximale Zeilenlänge 256 Zeichen

EBNF:

[ ]	...	optionale Angabe
{ }	...	Wiederholung, auch 0 mal
|	...	Alternative (xor)
abc	...	Nichtterminale
ABC	...	Terminale

Startsymbol: klassenbeschreibung bzw. instanzenbeschreibung

Produktionsregeln:

1) buchstabe = **a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z**

2) nichtnullziffer = **1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

3) ziffer = 0 | nichtnullziffer

4) sonderzeichen = **!** | **?** | **(** | **)** | **[** | **]** | **{** | **}** | **+** | **-** | **=** | **<** | **>** | **/** | **|** | **.** | **,** | **:** | **;** | ***** | **&** | **#** | **%** | **@** | **^** | **~** | Blank

5) nichtgrafischeszeichen = **\n** | **\t** | **\b** | **\r** | **\f** | **** | **\'** | **\0** | **\"**

6) zeichen = buchstabe | ziffer | sonderzeichen | nichtgrafischeszeichen

7) string = **"**{ zeichen } **"**

8) kommentar = { zeichen }

9) einsymbol = buchstabe { buchstabe | ziffer }

10) symbol = einsymbol | **UNKNOWN**

11) symbolliste = **BEGIN** { symbol } **END**

12) n_fachsymbol = symbol | symbolliste

13) zahl = nichtnullziffer { ziffer }

14) ganzzahl = **0** | zahl

15) gleitkomma = ganzzahl **.** [ganzzahl]

16) number = ganzzahl | gleitkomma

17) numberliste = **BEGIN** { number } **END**

18) nfachnumber = number | numberliste

19) constraintsdefinition = **CONSTRAINT** | **UNCONSTRAINT**

20) fuzzydefinition = **FUZZY** | **CRISP**

21) typbeschreibung = **NUMBER** | **SYMBOL** | **RELATION** | **BOOLEAN** | **STRING**

22) typdefinition = **TYPE** typbeschreibung

23) cardbeschreibung = zahl | **N**

24) carddefinition = **CARDINALITY** cardbeschreibung

25) symbolerlaubt = **ALLOWABLE_VALUES** symbolliste

26) relationerlaube **= ALLOWABLE_CLASSES** symbolliste

27) numberextrem = { **MINIMUM** number | **MAXIMUM** number }

28) numbererlaubt = **ALLOWABLE_VALUES** numberextrem

29) erlaubtdefinition = symbolerlaubt | relationerlaubt | numbererlaubt

30) einheitdefinition = **UNIT** string

31) vererbungbeschreibung = **NON** | **STANDARD** | **EXTENTABLE**

32) vererbungdefinition = **INHERIT** vererbungbeschreibung

33) attributdefinition = **DEFINE_ATTRIBUTE** symbol
[**SET_ATTRIBUTE_COMMENT** kommentar]
{ constraintdefinition | fuzzydefinition |
typdefinition | carddefinition |
erlaubtdefinition | einheitdefinition |
vererbungsdefinition }
END_DEFINE_ATTRIBUTE

34) numberwarnung = **WARNINGS** numberextrem

35) numberfehler = **ERROS** numberextrem

36) numberbelegung = **VALUE** nfachnumber
[**FUZZY_VALUE** numberextrem]
{ numberwarnung | numberfehler }

37) symrelextrem = { **MEMBER** symbolliste | **NOT_MEMBER** symbolliste }

38) symrelwarnung = **WARNINGS** symrelextrem

39) symrelfehler = **ERRORS** symrelextrem

40) symrelbelegung = **VALUE** nfachsymbol
[**FUZZY_VALUE** symbolliste]
{ symrelwarnung | symrelfehler }

41) stringliste = **BEGIN** { string } **END**

42) nfachstring = stringliste | string | **UNKNOWN**

43) stringbelegung = **VALUE** nfachstring

44) valuebelegung = numberbelegung | symbelegung | stringbelegung

45) attributbelegung = **SET_ATTRIBUTE** symbol valuebelegung
END_SET_ATTRIBUTE

46) einzelklasse = **DEFINE_CLASS** symbol
SUBCLASS_OF symbol
[**SET_CLASS_COMMENT** kommentar]
{ attributdefinition | attributbelegung }
END_DEFINE_CLASS

47) klassenbeschreibung = **KNOWLEDGE_TOPIC** symbol
KNOWLEDGE_DATE kommentar
[**SET_KNOWLEDGE_COMMENT** kommentar]
einzelklasse { einzelklasse }
END_KNOWLEDGE

48) einzelinstanz = **CREATE_INSTANCE** symbol
INSTANCE_OF symbolliste
[**SET_INSTANCE_COMMENT** kommentar]
{ attributbelegung }
END_CREATE_INSTANCE

49) instanzenbeschreibung = **PROJECT_TOPIC** symbol
PROJECT_DATE kommentar
[**SET_PROJECT_COMMENT** kommentar]
einzelinstanz { einzelinstanz }
END_PROJECT

Anmerkungen zur Syntax (Metaregeln):

10) symbol entspricht einem AutoLISP-Symbol. Zur Namensbildung und Eindeutigkeit gelten alle Einschränkungen, die für AutoLISP-Symbole gelten.

29) Ob die Regel für *symbolerlaubt, relationerlaubt* oder *numererlaubt* angewendet wird, hängt vom Typ des Attributes ab. Bei **BOOLEAN** wird ebenfalls *numbererlaubt* verwendet, wobei die Zahlen für die Beschreibung dann nur zwischen 0 (false) und 1 (true) liegen können. Unscharfe Boolean-Werte können durch *gleitkomma*-Zahlen abgebildet werden.

33) *constraintdefinition, fuzzydefinition, typdefinition, carddefinition, erlaubtdefinition* und *vererbungsdefinition* sind in beliebiger Reihenfolge aufführbar, müssen jedoch alle einmal und nur einmal angegeben werden.

36) *numberwarnung* und *numberfehler* sind in der Reihenfolge beliebig, müssen jedoch beide einmal angegeben werden, auch wenn sie nicht belegt sind.

40) *symrelwarnung* und *symrelfehler* sind in der Reihenfolge beliebig, müssen jedoch beide einmal angegeben werden.

36, 40, 43) Die Anzahl der Werte, die hinter **VALUE** aufgeführt werden, hängt von der Cardinalität ab. Wurde in der Attributdefinition als Cardinalität N angegeben, so sind durch **BEGIN** und **END** geklammerte beliebig lange Listen zu erwarten. Wurde eine konkrete Zahl angegeben, folgen entsprechend viele Werte zwischen **BEGIN** und **END**. Nur wenn als Cardinalität 1 vergeben wurde, erfolgt keine Blockung.

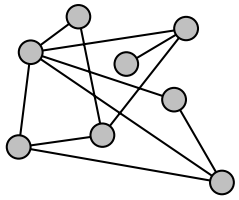
			NUMBER	SYMBOL	RELATION	BOOLEAN	STRING
UNCONSTRAINT	CRISP	1	■	■	■	■	■
		>= 2	■	■	■	■	■
		N	■	■	■		■
	FUZZY	1	■	■	■	■	
		>= 2	■	■	■	■	
		N	■	■	■		
CONSTRAINT	CRISP	1	■	■	■	■	
		>= 2	■	■	■	■	
		N	■	■	■		
	FUZZY	1	■	■	■	■	
		>= 2	■	■	■	■	
		N	■	■	■		

Tabelle C4-1: Attributtypen, ihre zulässigen Randbedingungen, Unschärfe, Cardinalität

Anhang D - Dokumentation NETGEN / ZGO (Version 3.0, Mai '95)

Autoren Dipl.-Ing. F. Steinmann
Dipl.-Inf. B. Brettschneider

Inhalt



1. Einführung

Beim Entwurf wird in frühen Phasen oftmals die Methode des funktionalen Entwerfens verwendet. Hiebei wird, nicht nur im Bereich der Architektur, das Artefakt spezifiziert durch

- **funktionstragende Objekte**, aus denen es besteht und
- **funktionale Beziehungen**, die zwischen den Objekten bei der Realisierung des Entwurfs gelten werden.

Das Ergebnis dieser Überlegungen sind i. allg. Listen von Objekten (Knoten) und Relationen (Kanten), die einen Graph bilden, d.h. genauer dessen Topologie. Diese Listen repräsentieren den Funktionsgraphen allerdings wenig anschaulich. Gesucht ist eine Graphendarstellung als 2D-Netzwerk aus Knotensymbolen (i. allg. Kreise) und Kanten (Linien), wie sie aus vielen Ingenieurbereichen bekannt sind, wie z.B.

- Funktionspläne in der Architektur
- Leitungsnetzpläne in Elektrotechnik und Elektronik
- Vorgangsnetze im Projektmanagement

1.1. Problemformalisierung

Gegeben ist der Graph **G** des Relationennetzwerkes. Die Objekte O_i stellen die Elemente der Knotenmenge $\{\mathbf{O}\}$ dar, die Relationen k_k werden durch die Kantenmenge $\{ k(O_i,O_j) \}$ gebildet. Gesucht werden die Positionen der Knoten $[x,y]_i^T$ in einer gegebenen Fläche, so daß **(I)** die Darstellungsfläche möglichst gleichmäßig mit Knoten gefüllt ist
(II) eine möglichst geringe Anzahl sich kreuzender Kanten entsteht.

Prinzipiell handelt es sich hier um eine Optimierungsaufgabe. Deren Lösung bereitet mit Standardmethoden allerdings Schwierigkeiten , weil

- es sich hier um eine Polyoptimierung handelt, deren Zielfunktionen (I, II) partiell konkurrieren sind (wie sich am Beispiel zeigen läßt)
- das Problem in die Klasse der gemischt ganzzahligen Optimierungen gehört
- eine mathematische Formulierung des Anspruchs (I) schwer fällt

1.2. Das Prinzip der sensorischen Karten

Das beschriebene Problem tritt in ähnlicher Form bei der Organisation des Nervensystems höherer Lebewesen auf. Die Zuordnung von Rezeptorzellen zu Zellarealen in der Großhirnrinde, in denen die Verarbeitung rezipierter Reize erfolgt, kann nicht genetisch fixiert sein, da dazu das Speichervermögen der DNS nicht ausreicht (/RITTER/). Diese Zuordnung muß in einem Selbstorganisationsprozeß erlernt werden.
Diese Abbildung ist nachbarschaftserhaltend, d.h. in der Rezeptorschicht benachbarte Neuronen sind mit Neuronen der Großhirnrinde 'verschaltet', die ebenfalls benachbart sind. Die Abbildung füllt weiterhin das Darstellungsgebiet ganz aus, d.h. es gibt keine Hirnareale, die funktionslos sind.

Diese Fähigkeit zur Selbstorganisation wird mit Neuronalen Netzen simuliert, wie sie von KOHONEN (u.a. in /MARTINEZ/) entwickelt werden. Sie enthalten folgende Vereinfachungen:

- Das Hirnareal wird durch eine Matrix **W** repräsentiert, wobei jedes Matrixelement $w_{i,j}$ ein Neuron darstellt
- Jedes Neuron $w_{i,j}$ speichert einen 2D-Vektor $[x,y]^T_{i,j}$, der die Stelle der Rezeptorfläche markiert, für die dieses Neuron am 'zuständigsten' ist.
- Die Rezeptorfläche hat die Größe $0<=x<=1$ und $0<=y<=1$.

Gesucht ist eine Belegung der Neuronen $w_{i,j}$ mit Vektoren $[x,y]^T_{i,j}$, so daß benachbarte Neuronen der Matrix benachbarte Vektoren speichern und die Rezeptorschicht gleichmäßig mit Vektoren überlagert ist (Abb. 1).

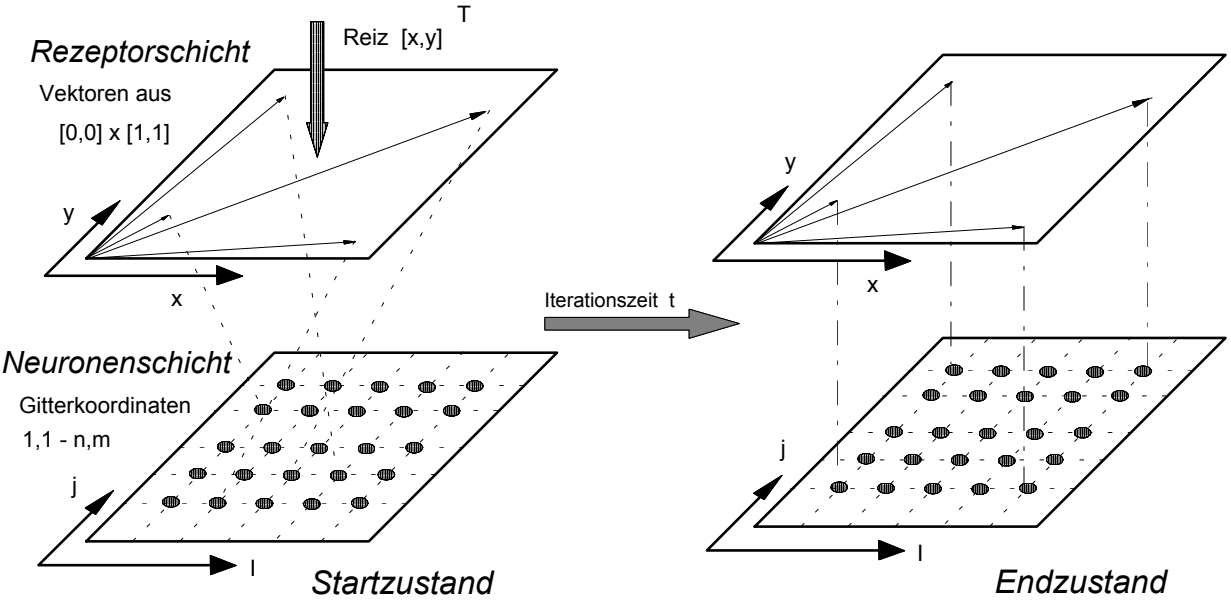


Abb. 1 Selbstorganisation in einem als Matrix verschaltete Netzwerk aus Neuronen $w_{i,j}$ die Vektoren $[x,y]^T_{i,j}$ beinhalten

Dieser Netztyp lernt 'nonsupervised', d.h. ohne Eingabemuster. Da nur bestimmte, zuständige Neuronen lernen dürfen, handelt es sich hier um 'competatives' Lernen (Kohonenregel).

Dazu wird nach folgendem **Algorithmus** verfahren :

- Initialisieren der Neuronen mit zufälligen Vektoren
- ***Wiederhole***

  - Zufälliges Aufbringen eines Reizes bei $[x,y]^T$ der Rezeptorfläche
  - Suche des Neurons $w_{i0,j0}$, das für diesen Reiz am 'zuständigsten' ist,

    d.h. für das der Abstand $|[x,y]^T_{Reiz} - [x,y]^T_{i0,j0}|$ minimal wird
  - Bestimmen der Neuronen $w_{i,j}$, die lernen dürfen. Für sie gilt, daß ihr Abstand e zum 'zuständigsten' Neuron $w_{io,jo}$ in Gittereinheiten kleiner als der aktuelle Lernradius $b(t)$ ist.

    Es gilt $d = (i-i0)^2 + (j-j0)^2 < b(t)$
  - Für alle Neuronen $w_{i,j}$, die ihren Zuständigkeitsvektor anpassen dürfen, wird $[x,y]^T_{i,j}$ in Reizes $[x,y]^T$ verschoben. Das Ausmaß der Verschiebung h nimmt mit der Entfernung e und mit zunehmender Iterationszeit t exponentiell ab.

    Es gilt die Vorschrift : $w_{i,j} := w_{i,j} + H[\, d\, ,\, t] * ([x,y]^T - w_{i,j})$

bis $t >= maxt$ oder $minh > ave = 1/s * \sum h(ti-s)...h(ti)$

Dabei gilt

$H(d,t) =$		$h0 * e(a0*t - d/b^2(t))$
$b(t)$	=	$b0 * e(-c0 * t)$
$a0$	=	$\ln(h0/h1) / maxt$
$c0$	=	$\ln(b0/b1) / maxt$
d	=	$(i-i0)^2 + (j-j0)^2$

Es bedeutet

t	t-ter Lernschritt
maxt	maximale Anzahl von Lernschritten
minh	Abbruchkriterium minimale Verschiebung
h0	anfänglicher Größenfaktor für die Verschiebung
b0	anfänglicher Lernradius (mitlernende Neuronen)
h1	Endwert des Größenfaktors für die Verschiebung
b1	Endwert des Lernradius (mitlernende Neuronen)
ave	Durchschnitt der letzten s Verschiebungen

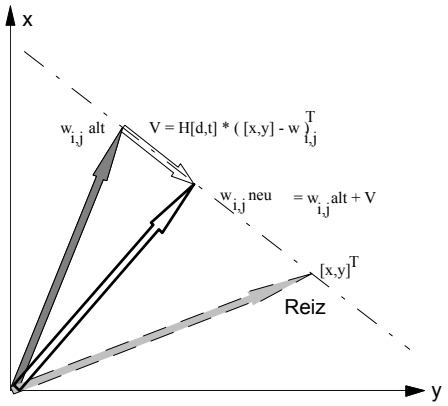


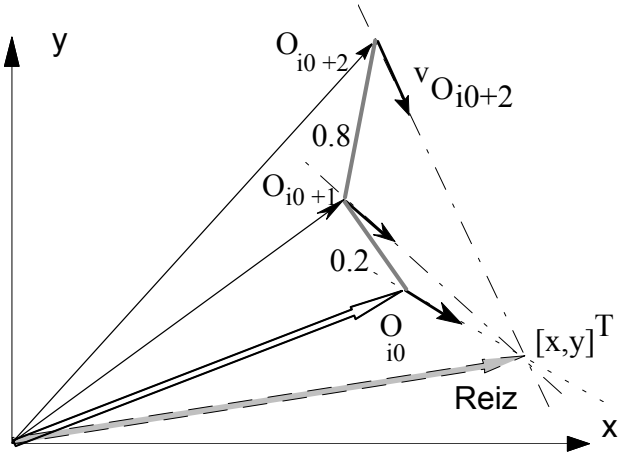
Abb. 2 Verschiebung des Zuständigkeits-vektors eines Neurons in einem Lernschritt

Das Verfahren läßt sich so verallgemeinern, daß beliebige n-dimensionale Regionen auf m-dimensionale Topologien abbildet werden, so beispielsweise die Knoten (Funktions-einheiten) in (Grundriß-) Polygonen zu konzentrieren. Auch das Arbeiten mit einem 2½D - Eingaberaum zur Abbildung von Gebäudekubaturen wird experimentiert und zeigt gute Ergebnisse. Hierbei werden die Funktionseinheiten in diskreten Sprüngen (Geschosse) im Schnittpolygon angeordnet. Durch Modifikation der Reizdichte läßt sich in bestimmten Gebieten eine höhere Neuronendichte erzielen. Das findet seine Analogie in der Natur, wo stark gereizte Areale (z.B. Fingerkuppen) über eine höhere Neuronenzahl in der Großhirnrinde verfügen.

1.5. Übertragung auf den Anwendungsfall

Bei der vorliegenden Anwendung treten i. allg. keine regelmäßigen Topologien wie Ringe, Gitter o.ä. auf, sondern beliebig strukturierte Graphen. Der Grundalgorithmus wurde dahingehend angepaßt, daß die Ermittlung der mitlernenden Neuronen durch Suche im Graphen mit der Suchtiefe des aktuellen Lernradius b(t) erfolgt. Dadurch ist die Darstellung gerichteter Relationen kein Problem mehr. Bei der Beschreibung von Funktionselementen in FUNPLAN ist es z.B. möglich, deren Lage im Grundriß (-polygon) anzugeben. (z.B. Knoten : 'Terrasse' Lage : (Süd, Südwest, West)). Der Algorithmus sollte diese Lagespezifikation beachten. Entsprechend der möglichen Werte für LAGE (9 festgelegte Symbole) wurde dazu die Reizfläche in 16 Teilflächen geteilt (siehe Parameter `#UNGEREIZT` hinten). Aus Flexibilitätsgründen werden die 9 Lagesymbole durch überlappende Teilflächen gebildet. Die Lagespezifikation wird auf zweierlei Arten verwendet :

- *Initialisierung der entspr. Knotenpositionen* beim Start des Algorithmus
- *Verbot von Knotenverschiebungen* aus den zugewiesenen Teilflächen



$h_{Oi} = fkt((1/n * \Sigma k_j) , b_{akt})$

j von i_0 bis n über

alle mitlernenden Knoten

Im Beispiel

$h_{Oi0+2} = fkt((0.2+0.8) / 2, b_{akt})$

Abb.3 Akkumulation von Kantenbewertungen

Die Einteilung dient ebenfalls zur groben Abbildung von Grundrißformen, in dem auf bestimmte Teilflächen keine Reizen aufgebracht werden und somit dort keine Knoten plaziert werden.

Eine weitere Anpassung besteht in der Einführung bewerteter Relationen. Die alle mitlernenden KnotenVerschiebungsfaktoren h werden dabei gemäß der akkumulierten Kantenbewertungen angepaßt. (Abb. 3). Die Wahl des Wertebereichs der Kantenbewertung bedarf noch weiterer Untersuchungen. Gegenwärtig werden die Bewertungen -, + unterstützt. Interessant ist hierbei im Beispiel, daß durch negative Knotenbewertungen auch Abstoßungen realisiert werden können.

2. Das Programm 'ZGO.EXE' (NETGEN)

Das vorliegende Programm kann überall da eingesetzt werden, wo beliebige Graphen visualisiert werden sollen. Die einfache Eingabe der Nachbarschaftsbeziehungen zwischen den einzelnen Knoten des Graphen genügt, um das Programm mit den notwendigen Informationen für die Visualisierung zu versorgen. Darüber hinaus können noch zusätzliche Angaben und Einschränkungen gemacht werden:

- Initialisierung von Knoten auf bestimmten Positionen
- Festhalten von Knoten in bestimmten Bereichen
- Sperren von Bereichen für die Belegung mit Knoten
- Wahl zwischen 1D, 1½D, 2D, 2½D, und 3D - Visualisierung
- Verstärken oder Abschwächen von Kanten

2.1 Hard- und Softwarevoraussetzungen

- **PC** mit mind. i80386- Prozessor und möglichst mit Coprozessor
- Betriebssystem MS-DOS
- Graphikkarte VGA
- mindestens 80 kByte freier Hauptspeicher
- mindestens 100 kByte Plattenspeicher (möglichst Festplatte)
- **UNIX**- Workstation
- einen C-Compiler (z.B. cc) zur Compilation des Quelltextes
- unter UNIX erfolgt keine Graphikausgabe (Weiterverarbeitung der Koordinatenausgabe mit einem Graphikprogramm)

2.2. Programmaufbau

Das Programm ist kommandozeilenorientiert, d.h. es erfolgt kein Dialog über Menüs. Um eine z.T. mehrzeilige Kommandoeingabe zu vermeiden, gibt es für die optionalen Parameter eine Konfigurationsdatei zgo.cfg, die mit einem beliebigen nichtformatierenden Editor (z.B. MS-DOS-edit bzw. UNIX-vi) verändert werden kann. Einziger Kommandozeilenparameter ist der Name der Datei mit den Angaben zu dem gesuchten Graphen. Die Ausgabe erfolgt generell in die Datei 'zgo.koo'. Bei Bedarf kann das Programm in ein Stapelverarbeitungsprogramm integriert und die Ausgabedatei anschließend umbenannt werden.

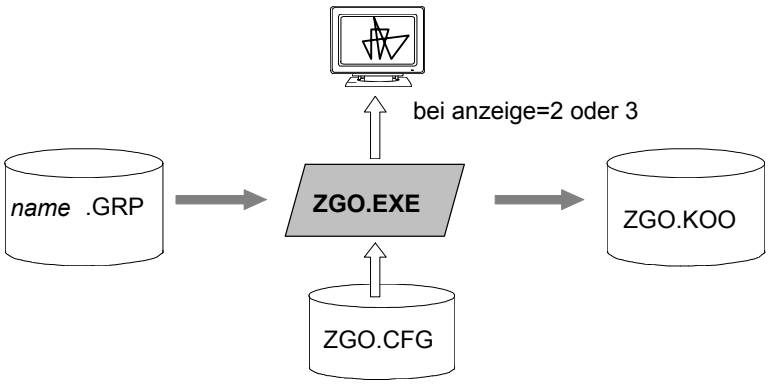


Abb. D1 Programmstruktur ZGO / NETGEN

2.3. Einführungsbeispiel

Gesucht ist die Anordnung von elektrischen Bauteilen auf einer Leiterplatte entsprechend einer gegebenen Schaltskizze. Zunächst überlegt man sich, daß die Bauteile zwei oder mehrere Anschlußpins haben und daß es entscheidend ist, welche Pins miteinander verbunden sind. Deshalb zerlegt man die Bauteile in ihre Pins. Die Bauteile repräsentiert man durch Kanten mit höherer Priorität. (`R`=Widerstand, `C`=Kondensator, `T`=Transistor, `L`=Lautsprecher). Man erstellt also folgende Datei *´el.grp´* (grp steht für Graph, diese Endung ist aber nicht bindend):

```
#KANTEN=UNGERICHTET
R1-1 R1-2 +
  :    :
C1-1 C1-2 +
  :    :
L-1 L-2 +
  :    :
T2-B T2-K +
T2-B T2-E +
```

Nun folgen die Angaben über Zusammenschaltung der Bauelemente untereinander und die Verbindungen mit den Anschlußklemmen `A` bis `I` :

```
A T2-E
T2-E R4-1
  :    :
R1-2 T1-B
T1-B C1-1
  :    :
R3-1 T1-K
T2-K L-1
```

Jetzt erfolgt die Angabe zur Dimensionalität des Graphen. Da eine Leiterplatte zweidimensional ist folgt: **`#ZWEIDIM`**
Für die Anschlußklemmen gibt es die Einschränkungen, daß die Klemmen `A` und `B` rechts oben (im Nordosten) der Leiterplatte liegen sollen, da sie mit dem Trafo verbunden werden sollen, die restlichen `C` bis `I` liegen auf der linken Seite (im Westen) und die Lautsprecheranschlüsse rechts unten (im Südosten).
Dazu gibt es das Schlüsselwort **`#HALTEN`** :

```
#HALTEN
A NO
B NO
...
I W
L-2 SO
```

Die letzte Einschränkung des Layouts besteht darin, daß sich in der rechten oberen Ecke der Trafo zur Stromversorgung befinden soll und in der rechten unteren Ecke der Lautsprecher. Deshalb dürfen sich an diesen Stellen keine Bauteile befinden. Dafür gibt es das Schlüsselwort **`#UNGEREIZT :`**

```
#UNGEREIZT
...X
....
....
...X
```

Das Ergebnis könnte z.B. so aussehen :

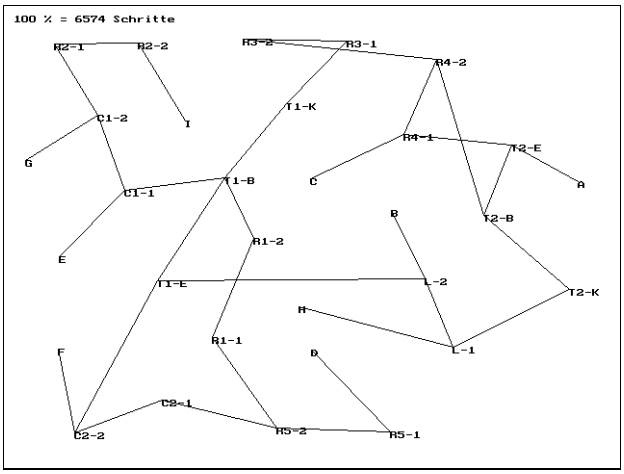


Abb D-0 gefundene Graphgeometrie des Beispielschaltkreises

2.4. Die Konfigurationsdatei '*ZGO.CFG*'

In der Konfigurationsdatei zgo.cfg steht jeder Parameter in einer neuen Zeile, nach dem letzten ist noch ein Zeilenumbruch anzufügen. In jeder Zeile steht zuerst ein Signalwort und dahinter eine Zahl, die durch ein Leerzeichen getrennt wird.
Kommentare können durch Beginnen einer neuen Zeile mit einem Nichtschlüsselwort eingefügt werden. Es folgt die Beschreibung der momentan unterstützten Signalwörter.

2.4.1. Parameter `init`

Mit **init** kann der Zufallsgenerator initialisiert werden. Dies kann genutzt werden, wenn immer die selbe Geometrie erzeugt werden soll, oder zu Testzwecken während der Implemenation. Es wird immer die gleiche Folge von Zufallszahlen generiert.

2.4.2. Parameter `Anzeige`

Mit dem Parameter ***Anzeige*** kann man, wenn die graphische Ausgabe möglich ist, diese steuern. Dabei sind folgende Zahlenwerte zulässig :

`0` ... es erfolgt keine graphische Ausgabe,
`1` ... es wird nur das Endergebnis graphisch dargestellt,
`2` ... es werden 20 Zwischenschritte und das Endergebnis dargestellt,
`3` ... wie bei 2, jedoch nach jeder Zwischendarstellung muß eine Taste gedrückt werden. Bei 1 bis 3 muß nach der Darstellung des Endergebnisses eine Taste gedrückt werden.

2.4.3. Parameter `beins, bnull`

Die Parameter **`bnull`** und **`beins`** beeinflussen den zeitlichen Verlauf der Anziehungskraft. Entscheidend ist das Verhältnis b_0/b_1:

Ist der Quotient $b_0/b_1>1$, so ist die Verlaufskurve mehr oder weniger nach unten durchgebogen, ist $b_0/b_1<1$ so ergibt sich eine Wölbung nach oben. Die Lage der Wölbung wird durch die Größe der Werte bestimmt.

Hier nun einige Beispiele für verschiedene Werte und ihre Auswirkungen auf die Verlaufskurve und den Verlauf der Visualisierung.

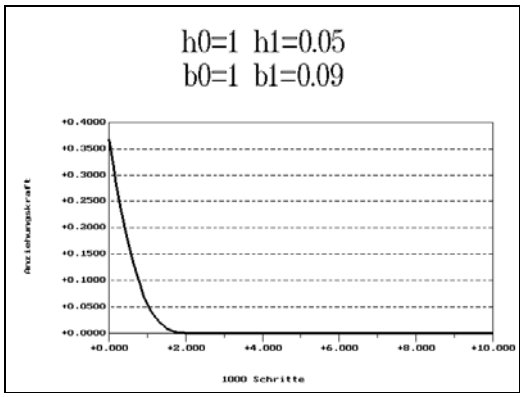


Abb D-1 Verlaufskurve Bsp. 1

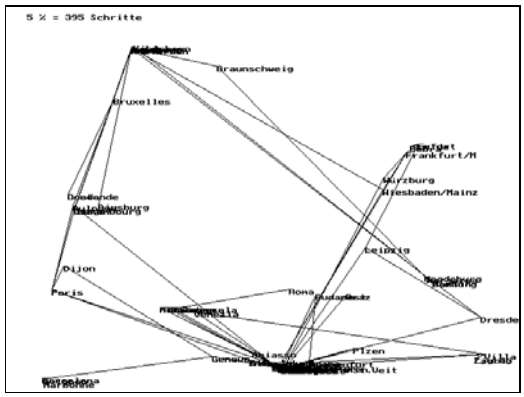


Abb D-2 Bsp. 1 nach 5% Bearbeitung

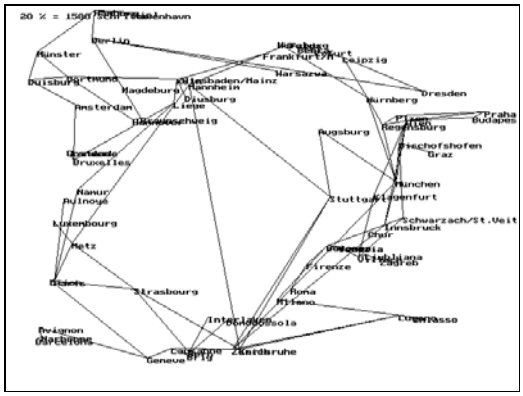


Abb D-3 Bsp. 1 nach 20% Bearbeitung

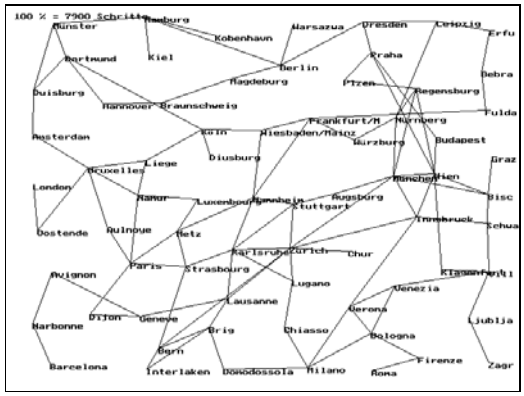


Abb D-5 Bsp. 1 nach 100% Bearbeitung

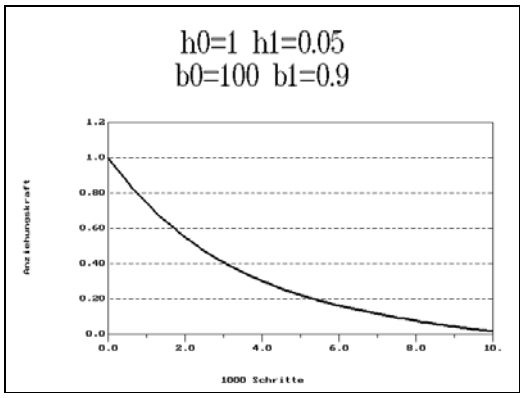


Abb D-6 Verlaufskurve Bsp. 2

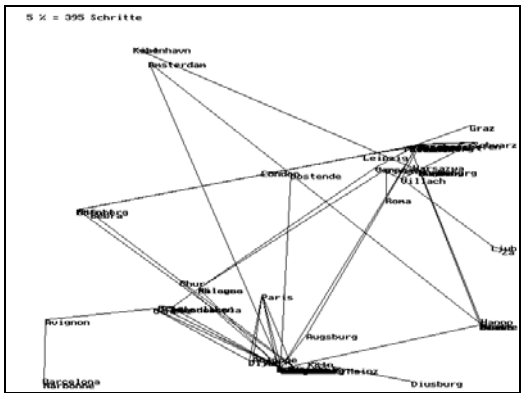


Abb D-7 Bsp. 2 nach 5% Bearbeitung

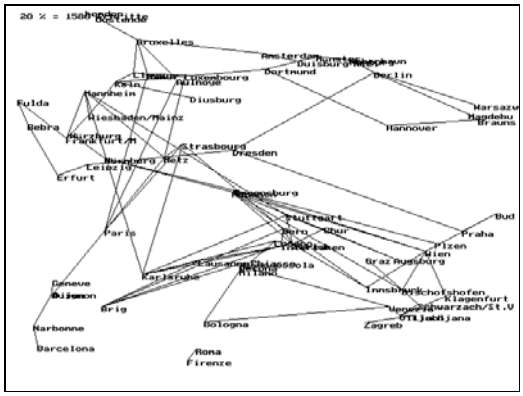


Abb D-8 Bsp. 2 nach 20% Bearbeitung

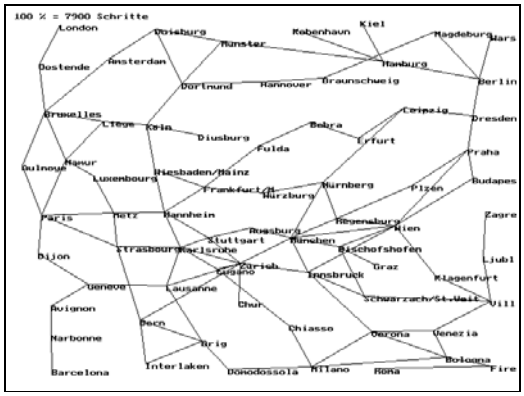


Abb D-10 Bsp. 2 nach 100% Bearbeitung

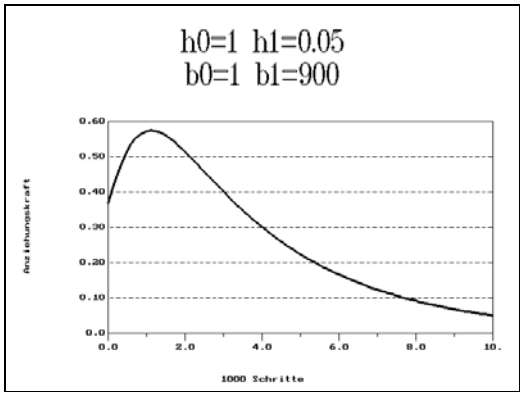


Abb D-11 Verlaufskurve Bsp. 3

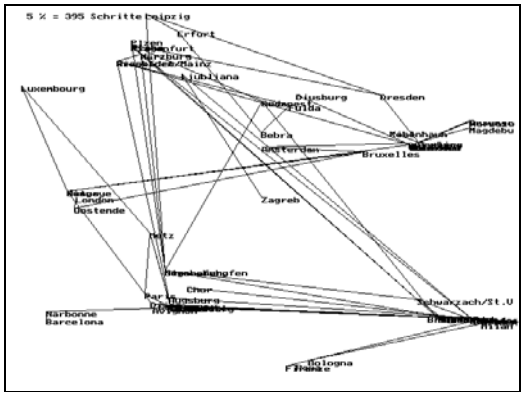


Abb D-12 Bsp. 3 nach 5% Bearbeitung

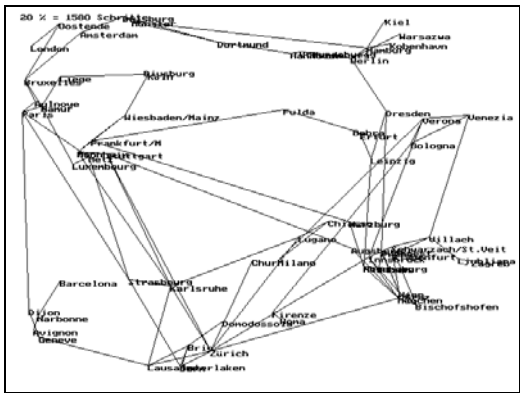


Abb D-13 Bsp. 3 nach 20% Bearbeitung

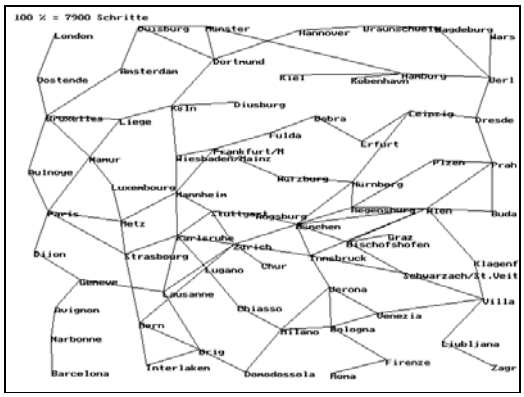


Abb D-15 Bsp. 3 nach 100% Bearbeitung

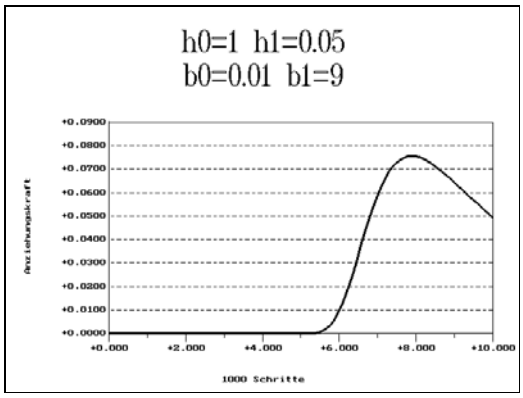


Abb D-16 Verlaufskurve Bsp. 4

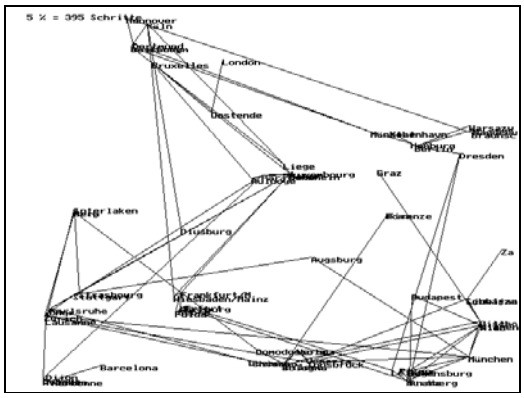


Abb D-17 Bsp. 4 nach 5% Bearbeitung

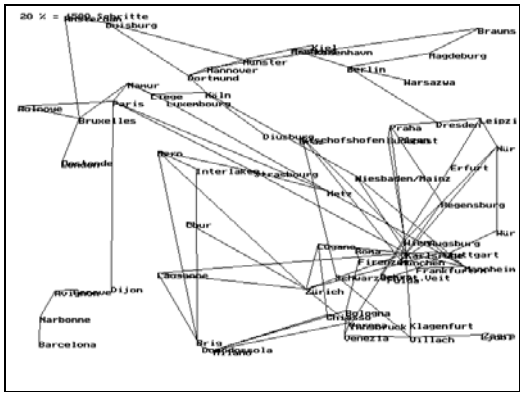


Abb D-18 Bsp. 4 nach 20% Bearbeitung

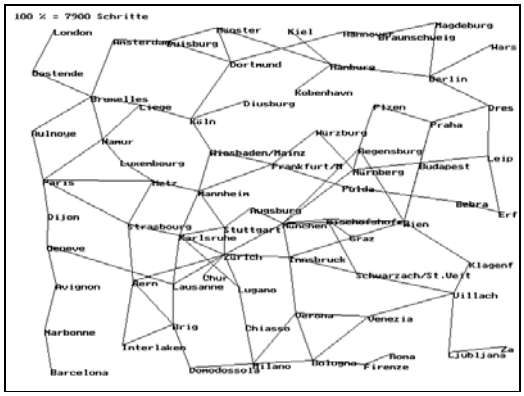


Abb D-20 Bsp. 4 nach 100% Bearbeitung

2.4.4. Parameter `heins, hnull`

Die Parameter **`hnull`** und **`heins`** bestimmen den Wert der Anziehungskraft zu Beginn (hnull) und am Ende (heins) der Bearbeitung des Graphen. Dabei können die Werte durch die Werte von bnull und beins ein wenig verfälscht werden, jedoch der Trend, ob die Anziehungskraft im Laufe der Bearbeitung verstärkt oder abgeschwächt werden soll, läßt sich auf jeden Fall bestimmen. Hierzu zwei Beispiele:

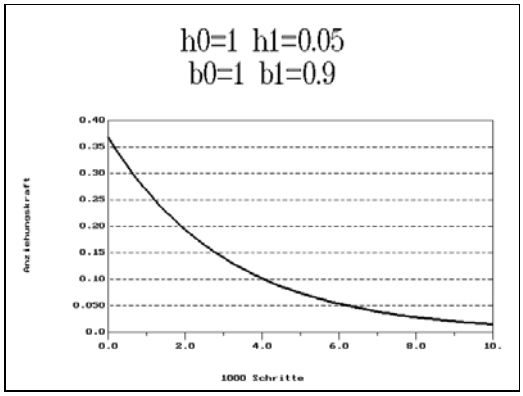


Abb D-21 Verlaufskurve Bsp. 5

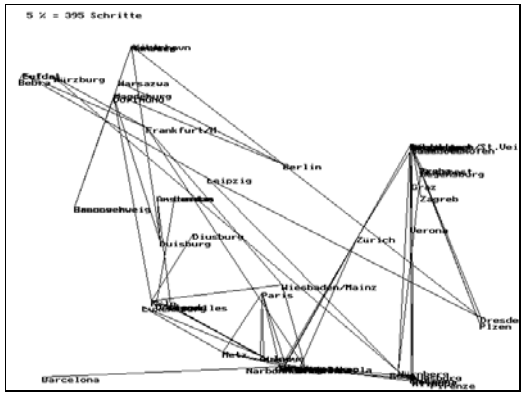


Abb D-22 Bsp. 5 nach 5% Bearbeitung

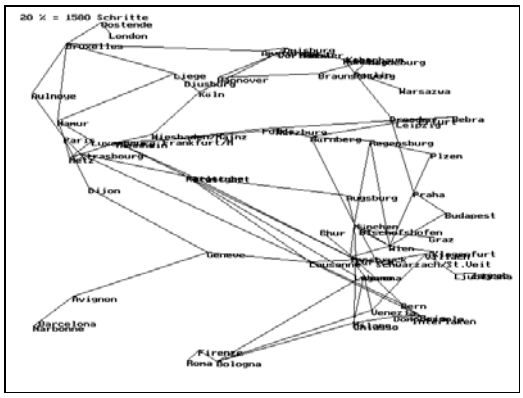


Abb D-23 Bsp. 5 nach 20% Bearbeitung

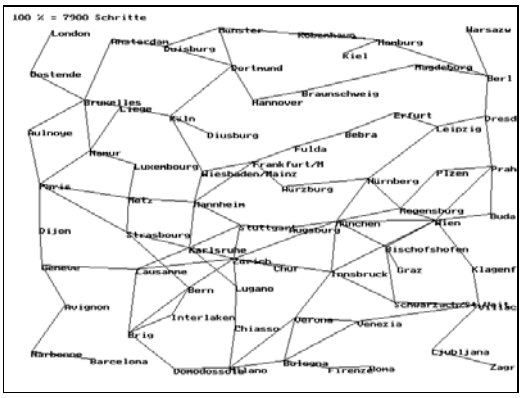


Abb D-25 Bsp. 5 nach 100% Bearbeitung

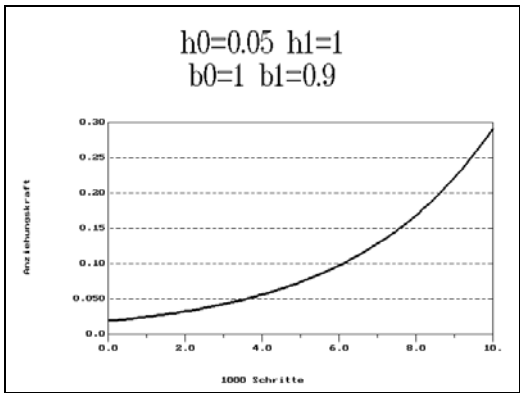


Abb D-26 Verlaufskurve Bsp. 6

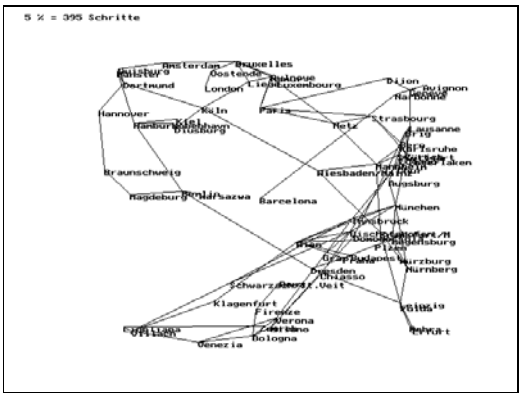


Abb D-27 Bsp. 6 nach 5% Bearbeitung

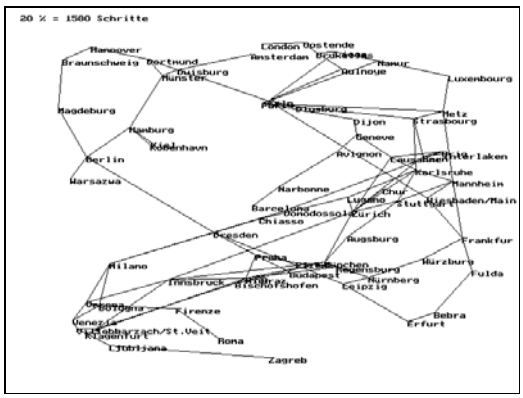


Abb D-28 Bsp. 6 nach 20% Bearbeitung

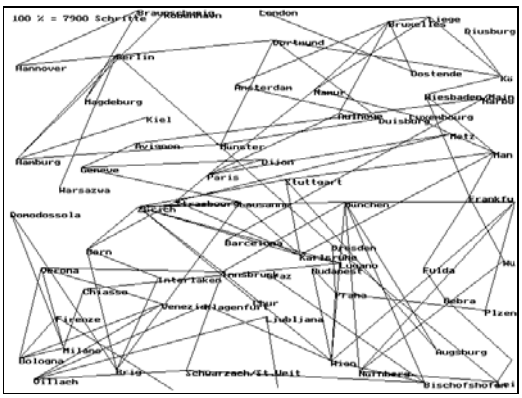


Abb D-30 Bsp. 6 nach 100% Bearbeitung

2.4.5. Parameter `maxlimit`

Mit **`maxlimit`** wird die maximale Anzahl von Reizen festgelegt, mit der der Graph bearbeitet wird. Dieser Parameter sollte in Abhängigkeit vom verwendeten Rechner und der maximal erwünschten Rechenzeit gewählt werden.

2.4.6. Parameter `maxreiz`

Mit dem Parameter **`maxreiz`** legt man die maximale Anzahl der Nachbarknoten fest, die bei einer Reizung mitgereizt werden. Dabei ist für normale Graphen maximal der Wert 3 zulässig, für Bäume höhere Werte. Eine Erhöhung dieses Wertes führt besonders in den ersten Schritten zu einer Verlangsamung und zu einer Konzentration der Knoten, was dann zu einer besseren Anordnung benachbarter Knoten führt. Zur Veranschauung hier der Vergleich zum Beispiel 5, bei dem der Parameter maxreiz 3 war, im Beispiel 7 ist er auf 1 gesetzt:

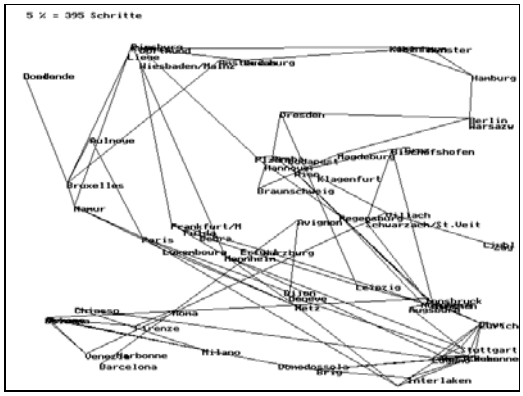


Abb D-31 Bsp. 7 nach 5% Bearbeitung

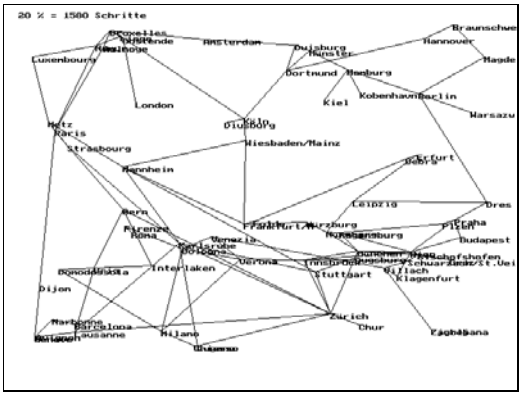


Abb D-32 Bsp. 7 nach 20% Bearbeitung

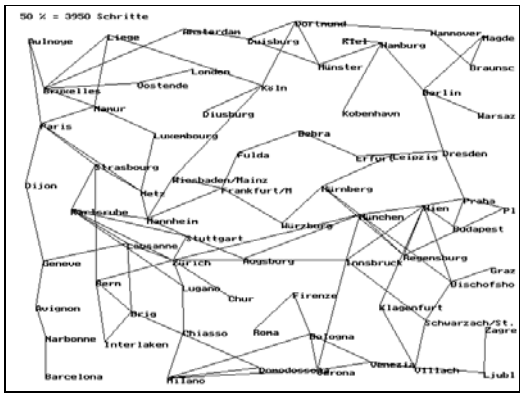


Abb D-33 Bsp. 7 nach 50% Bearbeitung

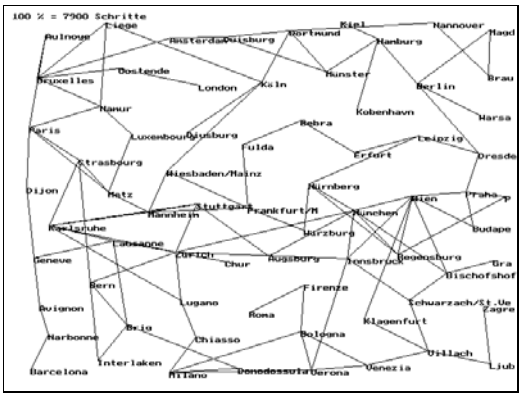


Abb D-34 Bsp. 7 nach 100% Bearbeitung

2.4.7. Parameter `mintime`

Mit **`mintime`** legt man die Mindestanzahl von Reizen fest mit der der Graph bearbeitet wird. Dieser Wert sollte ungefähr 10% von maxlimit betragen jedoch mindestens 1000.

2.5. Die Graphbeschreibungsdatei '*name.*GRP'

In den Graphbeschreibungsdateien stehen die Schlüsselwörter auf einer neuen Zeile, die mit dem Zeichen '**#**' beginnt. Schlüsselwörter müssen immer mit GROSZEN BUCHSTABEN und ohne Zwischenraum hinter das '**#**' geschrieben werden. Danach folgen die Daten, die durch das Schlüsselwort angekündigt wurden. Die Reihenfolge der einzelnen Abteilungen ist beliebig, sollten sich mehrere Angaben innerhalb einer Datei ausschließen, so gilt das zuletzt aufgeführte (z.B. **#ZWEIDIM** und **#DREIDIM**).
Kommentare beginne mit einem '**#**', gefolgt von einem Nichtschlüsselwort, das auf einer neuen Zeile eingefügt wird.

2.5.1. Beschreibung der Graphtopologie

Mit einem der oberen Schlüsselwörter

#KANTEN=GERICHTET oder
#KANTEN=UNGERICHTET

gibt man vor ob es sich um einen gerichteten oder ungerichteten Graphen handelt. Nach diesem Schlüsselwort folgt die Graphbeschreibung durch Angabe ihrer Kanten und damit implizit ihrer Knoten. Jede Kante steht auf einer neuen Zeile, es ist der Startknoten und der Endknoten der Kante, durch ein Leerzeichen getrennt, anzugeben. Zusätzlich kann durch die Anfügung durch ein '**+**' (durch ein Leerzeichen getrennt hinter dem Endknoten) die Kante verstärkt (= doppelte Anziehung) oder eines '**–**' die Kante abgeschwächt (= Abstoßung) werden.

2.5.2. Lageinitialisierung und Lagefixierung

Mit dem Schlüsselwort **HALTEN** können Knoten in bestimmten Bereichen festgehalten werden. Die Bereiche sind

NO ... Nordosten	**N** ... Norden	**NW** ... Nordwesten
O . . . Osten	**Z** ... Zentrum	**W** . . . Westen
SO ... Südosten	**S** ... Süden	**SW**...Südwesten

Die Abkürzung ist in einer Zeile hinter der Knotenbezeichnung, durch ein Leerzeichen getrennt, anzugeben. Jeder Knoten ist auf einer neuen Zeile aufzuführen. Meist genügt schon die Aufführung von wenigen Knoten in der Liste der Haltepunkte um dem Endgraphen eine Orientierung zu geben. Auch hier gilt, daß bei widersprüchlichen Angaben die letzte Angabe gilt (z.B. Zuordnung von einem Punkt zu verschiedenen Bereichen). Das bedeutet aber auch, daß keine Vereinigung verschiedener Bereiche möglich ist.

Bei der im Punkt 2.5.3 beschriebenen Bestimmung der Dimensionalität des Ausgaberaums läßt sich nach dem entsprechenden Schlüsselwort eine Lageinitialisierung von Knoten durchführen, z.B.

```
#DREIDIM
knoten12  0.2 0.4 0.9
```

Das hat nur bei geringen Startlernradien Bedeutung, kann dort aber zu Ergebnisverbesserung bei geringen Schrittzahlen eingesetzt werden.

2.5.3. Beschreibung des Ausgaberaumes

#EINDIM

Mit dem Schlüsselwort **#EINDIM** gibt man an, daß die Reizung auf einer Linie erfolgt und somit eine eindimensionale Abbildung entsteht. Außerdem ist es möglich, Knoten auf einer Position zu initialisieren oder auf einer Teilstrecke zu halten. Dazu schreibt man im Anschluß an das Signalwort für jede Knoten auf einer neuen Zeile den Knotenbezeichner und jeweils durch ein Leerzeichen getrennt die x- Koordinate (jeweils aus dem Bereich [0,1]).

#WURZEL

Diesem Schlüsselwort folgt in der nächsten Zeile nur der Knotenbezeichner der Wurzel eines Baumes und signalisiert dem Programm, daß der Graph als Baum zu behandeln ist. Dabei wird, ausgehend von der angegebenen Wurzel die Hierarchiestufe ermittelt und in der y-Koordinate fixiert. Dabei ist zu beachten, daß in einem Baum alle Kanten gerichtet sein müssen, da sonst keine Hierarchie ermittelt werden kann (= Ermittlung des Weges zur Wurzel) (Orientierung: von der Wurzel zu den Blättern). Sollte die Visualisierung ergeben, daß sich mehrere Knoten in der obersten Hierarchiestufe befinden, so kann dies an der falschen Orientierung einer oder mehrerer Kanten liegen oder der Graph ist nicht zusammenhängend.

#ZWEIDIM

Mit dem Schlüsselwort **#ZWEIDIM** gibt man an, daß die Reizung auf einer Fläche erfolgt und somit ein zweidimensionales Gebilde entsteht. Außerdem ist es möglich, Knoten auf diesem Quadrat zu initialisieren. Dazu schreibt man im Anschluß an das Signalwort für jeden Knoten auf einer neuen Zeile den Knotenbezeichner und jeweils durch ein Leerzeichen getrennt die x- und y- Koordinate (jeweils aus dem Bereich [0,1]).

#ZWEIEINHALBDIM

Mit dem Schlüsselwort **#ZWEIEINHALBDIM** lassen sich dreidimensionale Graphen erzeugen, bei denen den Knoten die z-Koordinate fest vorgegeben wird. Hierzu schreibt man nach diesem Schlüsselwort jeden Knoten auf eine neue Zeile und dahinter durch ein Leerzeichen getrennt die z-Koordinate aus dem Bereich

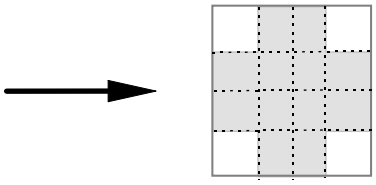
#DREIDIM

Mit dem Schlüsselwort #**DREIDIM** gibt man an, daß die Reizung nicht auf einer Fläche, sondern innerhalb eines Würfels erfolgt und somit ein dreidimensionales Gebilde entsteht. Außerdem ist es möglich Knoten in diesem Würfel zu initialisieren. Dazu schreibt man im Anschluß an das Signalwort für jeden Knoten auf einer neuen Zeile den Knotenbezeichner und jeweils durch ein Leerzeichen getrennt die x-, y- und z- Koordinate (jeweils aus dem Bereich [0,1]).

#UNGEREIZT

Nach diesem Schlüsselwort **#UNGEREIZT** können Bereiche angegeben werden, die ungereizt bleiben sollen, was zur Folge hat, daß in diesen Bereichen sich keine Knoten befinden werden. Hierzu denkt man sich die Fläche in ein 4x4-Raster aufgeteilt, so daß 16 Teilbereiche entstehen. Hinter diesem Schlüsselwort gibt man diese Raster an und markiert alle nicht zu reizenden Teilflächen mit '**X**', die zu reizenden Teilflächen dagegen mit '**.**' , d.h. es müssen alle 16 Teilbereiche gemäß ihrer Lage angegeben sein:

```
#UNGEREIZT
X . . X
. . . .
. . . .
X . . X
```



läßt die Ecken ungereizt. Für die zu reizenden Flächen kann ein beliebiges Zeichen außer `'X'` verwendet werden z.B. '.'. Die Einschränkung der Reizfläche wird nur berücksichtigt, wenn das Raster vollständig in dieser vierzeiligen Form angegeben wird.
Wird eine **#EINDIM** - Abbildung verwendet, ist nur eine Zeile anzugeben. Wird eine **#ZWEIEINHALBDIM** oder **#DREIDIM** - Abbildung verwendet, wird das Schnittpolygon in der x-y Ebene beschrieben. Die Höhe des Körpers ist 1.

3. Bewertung des Verfahrens

Mit dem vorgestellten Verfahren gelingt die übersichtliche Visualisierung von Funktionsplänen. Selbst größere Netze lassen sich mit akzeptablen Rechenzeiten visualisieren.
Der Algorithmus kann durch die Vorgabe der Parameter `b0 , h0, maxt, minh, h1` und `b1` in der Konfigurationsdatei ´ZGO.CFG´ der Netzgröße flexibel angepaßt werden. Da das Programm z.Zt. Daten aus Dateien liest und schreibt lassen sich Schnittstellen leicht gestalten. Für die 3D Visualisierung in AutoCAD lassen sich so beliebige Blöcke als Knotensymbole mit beliebiger Attributierung (z.B. zur Knotenbeschriftung) wählen. Ein entsprechendes Programm auf der Basis des AutoLisp Objektsystemes liegt vor. ´**ZGO**´ wird z.Z. auch im System FUNPLAN zur Unterstützung des frühen architektonischen Entwurfs unter MSWindows © als Hintergrundprozeß eingebunden.

4. Literatur

/HAUGK/ Haugk,S. : "Untersuchung über die Anwendbarkeit Neuronaler Netz", Diplomarbeit, Weimar 1990

/KOHONEN/ Kohonen,T. ; Ritter,H. : "Selforganizing semantic maps", Biological Cybernetics Nr. 61 , 1989

/KRUSE/ Kruse,H.; Mangold,R. u.a. : "Programmierung Neuronaler Netze", Addison Wesley, Bonn 1991

/MARTINEZ/ Martinez,T.;Ritter,H.; Schulten,K.: "Neuronale Netze Einführung in die Neuroinformatik selbstorganisierender Netzwerke", Addison Wesley, Bonn 1990

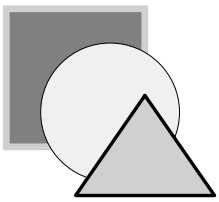
/RITTER/ Ritter,H. : Kursscript "Neuronale Informationsverarbeitung und Robotik", KIFS 1993 , Günne/ Uni Bielefeld 1993

/STEINMANN/ Steinmann, F.: „Die Anwendung des Prinzips der sensorischen Karten zur Visualisierung komplexer Beziehungsnetze", Tagungsband 14.IKM , HAB Weimar, März 1994

Anhang E - Dokumentation A L O S Vers. 95c (Okt. '95)

Autor : Dipl.-Ing F.Steinmann

Inhalt



1. Zielstellung

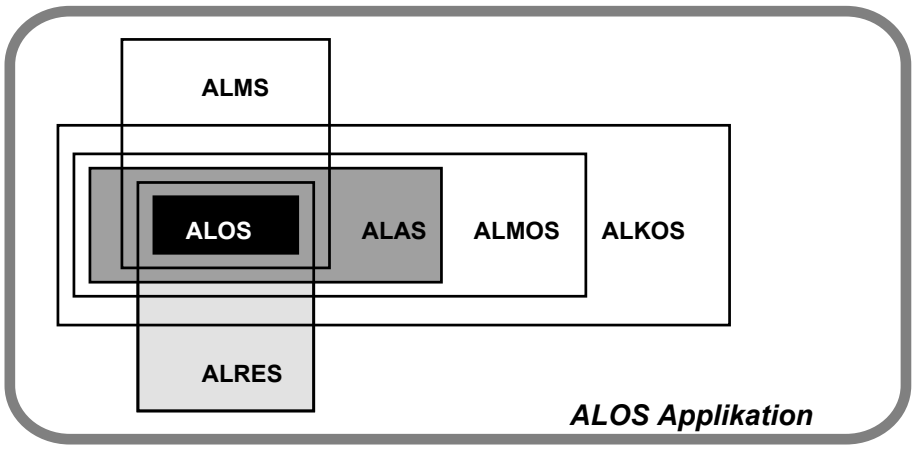
Objektorientierte Denk- und Arbeitsweisen werden durch die Unterstützung der objektorientierten Programmierung in LISP schon lange gepflegt. Kommerzielle CommonLisp-Systeme (z.B. Lucid- CommonLisp oder Allegro-CommonLisp) beinhalten schon seit langem das standardisierte Objektsystem **CLOS** (CommonLisp Objekt System) und Flavors (System zum Nachrichtenaustausch). Selbst kleine Public Domain Produkte wie XLisp (D.Baetz) beinhalten ein Objektsystem. Der bevorzugte Einsatz von Lisp für Zwecke der KI fördert diesen Trend noch.
Mit den Frames ist eine Wissensrepräsentation gegeben, die selbst objektorientiert ist, in einigen Punkten die Mächtigkeit der OOP aber noch übertrifft (Facetts). Erste Implementationen von Frames waren Lisp-Implementationen.
Es ist vor allem die Gleichbehandlung von Daten und Programmen (syntaktisch wie speichertechnisch) sowie die sehr einfache Syntax der Sprache, die Lisp für solche Aufgaben prädestiniert.
Mit dem CLOS-System als Teil des Sprachstandards CommonLisp ist ein effizientes Werkzeug zur objektorientierten Programmierung gegeben, da alle entsprechenden Funktionen 'built-in' sind, also maschinennah effizient programmiert sind. Die Standardisierung führt zu portablen, gut lesbaren Programmen oder zumindest sollte sie das.
Diese Implementation in built-in Funktionen bringt aber auch Nachteile. Es ist somit nicht möglich, die Funktionalität des Objektsystems zu ändern oder zu erweitern wie beispielweise Aggregations- und Analysefunktionen, Facettisierungen von Attributen und Methoden und gesteuerte Vererbung. Die Gestaltung von Schnittstellen ist schwierig. Der CLOS-Standard steht zudem in kleineren Objeksystemen wie XLisp oder AutoLisp nicht zur Verfügung. Gerade für CAD-Prozesse ist aber die Sicht der Objektorientiertheit die gegebene. Es sollte daher auch für AutoLisp als der momentan meistgenutzten Programmierschnittstelle der Standard-CAD-Software AutoCAD ein intuitives, dem CAD-Prozeß angepaßtes (Aggregieren!) Lisp-Objektsystem zur Verfügung stehen. Es wäre Grundlage für das Experimentieren objektorientierter Modellierungstechniken in AutoCAD.

Wesentliche Merkmale von ALOS (Basismodul) sind:

- Offener in AutoLisp implementierter Objektaufsatz
- Hierarchisches Klassensystem mit einfacher Vererbung
  (statisch und dynamisch beim Erzeugen)
- Laufzeitdynamische Objektstrukturen
- Funktionen zum Lesen und Schreiben von Objekten, Komplexobjekten sowie Taxonomien

- Angebot an Analysefunktionen für Taxonomien und Komplexobjekte zur Unterstützung der Programmierung

Alle Funktionen können als Funktionen direkt genutzt werden, sollten aber auch als Methoden eines Objekts nutzbar sein, d.h. sie sind lokal in ihrer Funktionalität, können verändert werden. Aus Gründen der begrenzten Ressourcen unter AutoLisp und mangelnder Effizienz wurde jedoch vorerst darauf verzichtet.
ALOS ist der Basismodul auf den die Teilsysteme zur Aggregationsunterstützung **ALAS** sowie der vorwärtsverkettende Regelinterpreter **ALRES** aufsetzen.
Konzipiert sind weiterhin die Module zur Entwurfssteuerung **ALMOS**, zur Konfiguration **ALKOS**, zum Monitoring **ALMS** (Teile schon in **ALOS** enthalten), diese sind allerdings gegenwärtig nur in Xlisp implementiert und ihre Übertragung nach AutoLisp ist fraglich.



Obwohl AutoLisp eher als Makrosprache mit nur geringer Programmierfunktionalität konzipiert war, ist ihr Grundkonzept so tragfähig, daß selbst komplexe Systeme mit ihr entwickelt wurden. Viele Grundfähigkeiten von Standard-Lispdialekten vermißt man jedoch schmerzlich, wie z.B. ‘unbound’ für Symbole, des weitern Datentypen (arrays, records, streams), Property-Listen, Zellstruktur von Symbolen, Makros.
Die oftmals mangelnde Expressivität der ALOS Funktion ist z.T. ebenfalls in der Sprache AutoLisp begründet, die leider keine optionalen Parameter kennt! Die Effizienz des Interpreters läßt zu wünschen übrig. Die Verwendung eines AutoLisp Compilers ist möglich und bringt Verbesserungen bringen (ca. 5% - 10% Laufzeitverbesserungen). Bei dessen Verwendung werden entwickelte Applikationen durch die binäre Verschlüsselung des Compilers schützbar.

Die Beispiele der Funktionsnutzung beziehen sich auf das Beispiel 'FLAECHEN.LSP' , weiterhin sei auf das Demo 'KRANWELT.LSP' verwiesen.

Bei **ALOS** handelt es sich um ein Experimentalsystem, das als Lehrsoftware zur Implementation von OO-Techniken in XLisp entwickelt und später nach AutoLisp portiert wurde.
Sowohl dieses Manual als auch die letzte Fassung von **ALOS** selbst wurden (wie immer) unter Zeitdruck erarbeitet, was Fehler und didaktische Schwächen zur Folge gehabt haben dürfte. Für Hinweise zu Fehlern und für inhaltliche Verbesserungsvorschläge ist der Autor dankbar. Die Software versteht sich als 'public domain' und unterliegt den dafür gültigen rechtlichen Normen.

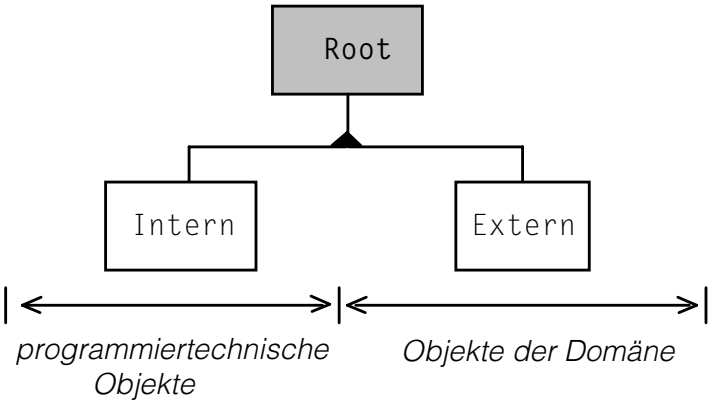
2. ALOS - Basismodul

2.1 Datenstrukturen des Objektsystems

Repräsentant eines Objektes ist ein Symbol. Das allgemeinste Objekt heißt **Root**. Namenssymbole dürfen im weiteren nicht mehr anderweitig verwendet werden. Das Namenssymbol enthält als Wert eine ASSOC-Liste aller mit dem Objekt assoziierten Informationen, d.h. Daten und Methoden in Form von LAMBDA-Ausdrücken. Alle Elemente der ASSOC-Liste (Schlüssel-Wert Paare) werden als Slots bezeichnet.
Die ASSOC-Liste ist wie folgt strukturiert (lokal verfügbare Slots):

	Klasse	**Instanz**
Systemdefiniert (ab **ROOT**)	**(** (TYP KLASSE) (SUPER *superobjname*) (SUB (*subobj1 subobj2 ...*)) (INSTANZEN (*inst1 inst2 ...*))	**(** (TYP INSTANZ) (SUPER *superobjname*)
Systemdefiniert (ab **EXTERN**)	(NAME *namestring*) (PART-OF (*komplex1 komplex2* *...*)) (HAS-PART (*part1 part2...*)) (RULES (*rule1 rule2 ...*))	
Nutzerdefiniert	(*slotname_1* *value*) .... (s*lotname_n* *value*) **)**	

Das ALOS-Objektsystem besteht initial aus den Klassen



Prinzipiell verfügt **ALOS** über eine dynamische Vererbung, d.h. es wird zur Laufzeit ermittelt, von wo eine Slotinformation bezogen wird. Zusätzlich können aber Klassen und Instanzen statisch **erzeugt** werden. Dabei werden schon beim Anlegen des Objektes alle über Erbschaft verfügbaren Slots physisch in die Objektbeschreibung (ASSOC-Liste) eingetragen. Treten dann allerdings dynamische Veränderungen der Slotbelegung der entsprechenden Slots in den Superklassen auf, werden diese Veränderungen nicht mehr weitergereicht, da die Werte in den statischen erzeugten Objekten lokal sind.

In **ALOS** verfügen Instanzen (und nur die) über Objektidentität, d.h. es können Instanzen mit gleichem Namen(-sstring) mehrfach auftreten, da die Instanzinformationen (ASSOC-Liste) in intern generierten Symbolen (`$IAAAA ... $IXXX`) abgelegt werden. Zum Zugriff über den Namen(-sstring) wird im Symbol $ITAB0 eine dynamische Instanztabelle geführt (siehe Modul *AOITAB.LSP*).

Attribut- und Methodendefinition ist nur bei Klassen zulässig. Methoden werden als Funktionen mit internem Namen abgelegt (`$MAAAA ... $XXXX`). In den Methodenslots der Objekte werden LAMBDA-Ausdrücke generiert, die lediglich den Aufruf der entsprechenden internen Funktion realisieren. Die Argumentliste wird standardmäßig um das Argument `SELF` verlängert, das beim Aufruf an das Objektsymbol gebunden ist, welches die Nachricht empfängt.

Zur Nutzung des ALSO-Basismodules muß das File '*ALOS.LSP*' geladen werden, das selbst wiederum für das Laden aller weiteren erforderlichen Files des Basismodules sorgt.

2.3. Modulbeschreibung

Alle Funktionen mit 'kryptischer' Namensgebung sind Kurzformen von Funktionen, ohne jegliche Fehlertests.

Modul : **ALOS.LSP**
Beschreibung : Der Modul lädt automatisch alle Teile des ALOS-Grundmoduls

---

Modul : **AOINIT.LSP**
Beschreibung : Der Modul definiert die Basisobjekte sowie deren Struktur und initialisiert Systemvariable (beginnen mit $.....)

```
Exportierte Objekte :  Klassen  ROOT, INTERN, EXTERN
                       Slots bei ROOT:  TYP, SUPER, SUB, INST
                       Slots bei EXTERN: HAS-PART, PART-OF, RULES, NAME
                       Symbole  NICHTS, $INST0, $MET0, $SYM0, $ITAB0
Exportierte Funktionen :
      (DEFUN ResetALOS ()  ...  )
```

Modul : **AOERZEUG.LSP**
Beschreibung : Der Modul enthält alle Funktionen zum Erzeugen von Klassen, Instanzen sowie deren Slots und der Klassenmonitore

```
Exportierte Funktionen :
      (DEFUN ErzeugeKlasse (class super) ... )
      (DEFUN ErzeugeStatischeKlasse (class super) ... )
      (DEFUN ErzeugeInstanz (namestring super) ... )
      (DEFUN SEI (super) ... )
      (DEFUN ErzeugeStatischeInstanz (name super) ... )
      (DEFUN KopiereInstanz (inst) ... )
      (DEFUN SKI (inst) ... )
      (DEFUN ErzeugeAttribut (class attr init) ... )
      (DEFUN ErzeugeMethode (class met paramlist metkoerperlist) ... )
```

Modul : **AOMONITO.LSP**
Beschreibung : Der Modul enthält die Funktion zum Erzeugen folgender Klassenmonitoren

a) BEIERZEUGEN, BEIEINBAU,
b) BEILOESCHEN, BEIAUSBAU

Die Monitore feuern nach der Aktion der sich aus ihrem Namen ergibt.

```
Exportierte Funktionen :
    (DEFUN ErzeugeKlassenMonitor (class monitor locsymli metkoerperlist)
    ... )
```

Modul : **AOLOESCH.LSP**
Beschreibung : Der Modul enthält alle Funktionen zum Loeschen von Klassen, Instanzen sowie deren Slots

```
Exportierte Funktionen :  (DEFUN LoescheInstanz (inst) ... )
                          (DEFUN SLOI (inst) ... )
                          (DEFUN LoescheLokInstanzen (class) ... )
                          (DEFUN LoescheInstanzen (class) ... )
                          (DEFUN LoescheKlasse (class) ... )
                          (DEFUN LoescheKlasseGanz (class) ... )
                          (DEFUN LokLoescheSlot (obj slot) ... )
                          (DEFUN SLLOS (obj slot) ... )
                          (DEFUN LoescheSlot (obj slot) ... )
                          (DEFUN SLOS (obj slot) ... )
```

Modul : **AOSENDE.LSP**
Beschreibung : Der Modul enthält die Funktionen zum Versenden und Auswerten von Nachrichten an Klassen und Instanzen

```
Exportierte Funktionen :  (DEFUN SendeNachricht (obj msg paramlist) ... )
                          (DEFUN SN (obj msg paramlist) ... )
```

Modul : **AOAENDER.LSP**
Beschreibung : Der Modul enthält alle Funktionen zum Aendern der Slots von Klassen und Instanzen

```
Exportierte Funktionen :
            (DEFUN LokAendereAttribut (obj attr new) ... )
            (DEFUN SLAe (obj attr new) ... )
            (DEFUN GlobalAendereAttribut (obj attr new) ... )
            (DEFUN SGAe (obj attr new) ... )
            (DEFUN SetO (obj attr new) ... )
            (DEFUN AendereSuper (obj newsuperclass) ... )
            (DEFUN AendereName (obj newnamestring) ... )
            (DEFUN SAN (obj newnamestring) ... )
```

Modul : **AOPRED.LSP**
Beschreibung : Der Modul enthält prädikative Funktionen zur Analyse des ALOS Objektsystemzustandes

```
Exportierte Funktionen :
         (DEFUN ALOSObjekt? (obj) ... )
         (DEFUN ALOSObj? (obj) ... )
         (DEFUN OTyp  (obj) ... )
         (DEFUN Klasse?  (obj) ... )
         (DEFUN Instanz? (obj) ... )
         (DEFUN Slot?  (obj slot) ... )
         (DEFUN SS?  (obj slot) ... )
         (DEFUN LokSlot?  (obj slot) ... )
         (DEFUN SLS?  (obj slot) ... )
         (DEFUN Methode?  (obj met) ... )
         (DEFUN LokMethode?  (obj met) ... )
         (DEFUN SLM?  (obj met) ... )
```

Modul : **AOANALY1.LSP**
Beschreibung : Der Modul analysiert die Struktur von Objekten

```
Exportierte Funktionen :
          (DEFUN LokMethoden (obj) ... )
          (DEFUN SLM (obj) ... )
          (DEFUN Methoden (obj) ... )
          (DEFUN SM (obj) ... )
          (DEFUN LokAttribute (obj) ... )
          (DEFUN Attribute (obj) ... )
          (DEFUN LokSlots (obj) ... )
          (DEFUN SLS (obj) ... )
          (DEFUN Slots (obj) ... )
          (DEFUN SS (obj) ... )
          (DEFUN WoSlot (obj slot) ... )
          (DEFUN SWS (obj slot) ... )
          (DEFUN WoSlotDefinition (obj slot) ... )
```

Modul : **AOANALY2.LSP**
Beschreibung : Der Modul enthält alle Funktionen zur Analyse der Struktur der Taxonomie, d.h. der Klassen, Instanzen.

```
Exportierte Funktionen :
          (DEFUN Vorgaenger (obj) ... )
          (DEFUN SV (obj) ... )
          (DEFUN Nachfolger (class) ... )
          (DEFUN SNF (class) ... )
          (DEFUN NachfolgerKlassen (class) ... )
          (DEFUN LokNachfolgerKlassen (class) ...)
          (DEFUN SLNKL (class) ...)
          (DEFUN Instanzen (class) ... )
          (DEFUN SI (class) ... )
          (DEFUN LokInstanzen (class) ...)
          (DEFUN SLI (class) ...)
          (DEFUN OberKlasse (obj1 obj2) ... )
          (DEFUN HoleNamen  (ObjSym) ... )
          (DEFUN SHN  (InstSym) ... )
          (DEFUN HoleInstanzSymol (InstNameString) ... )
          (DEFUN SHSYM (InstName) ... )
```

Modul : **AOZUGRI.LSP**
Beschreibung : Der Modul enthält alle Funktionen zum (reinen) Zugriff auf Klassen, Instanzen, d.h. deren Slots

```
Exportierte Funktionen :
            (DEFUN Hole (obj slot ) ... )
            (DEFUN SH (obj slot ) ... )
            (DEFUN LokHole (obj slot) ... )
            (DEFUN SLH (obj slot) ... )
```

Modul : **AOLASI.LSP**
Beschreibung : Der Modul enthält Funktionen zum Laden und Sichern des kompletten aktuellen Objektsystems. Beim Laden wird der aktuelle Sytemzustand überschrieben, Symbolspeicher für Instanzen und Methoden werden zurückgesetzt.

```
Exportierte Funktionen :
          (DEFUN SichereALOS (outdatei) ... )
          (DEFUN LadeALOS (indatei) ... )
```

Modul : **AOZEIG.LSP**
Beschreibung : Der Modul enthält alphanum. Anzeigefunktionen (Darstellen eines Objektbaumes) ...)

```
Exportierte Funktionen :
           (DEFUN ZeigeObjekt (obj) ... )
           (DEFUN SZO (obj) ... )
           (DEFUN ZeigeObjektGanz (obj) ... )
           (DEFUN SZOG (obj) ... )
           (DEFUN ZeigeHierarchie (topclass) ... )
           (DEFUN SZH (topclass) ... )
           (Defun ZeigeAttribute (obj) ... )
           (Defun SZA (obj) ... )
           (Defun ZeigeAttributeGanz (obj) ... )
           (Defun SZAG (obj) ... )
```

Modul : **AOERROR.LSP**
Beschreibung : Der Modul enthält die Fehlerfunktion

```
Exportierte Funktionen :
         (DEFUN ObjErr (errobj errfun paramlist) ... )
```

Modul : **AOITAB.LSP**
Beschreibung : Der Modul enthält Funktionen zur Verwaltung von Informationen in dyn. Tabellen als ASSOC-Listen folgender Struktur :

```
(  (key1  (ele11 ele12 ... ele1n1)            ← Col 1
   (key2  (ele21 ele22 ... ele2n2)    ....          ...
   (keym  (elem1 elem2 ... elemnm)       ← Col m
)
Exportierte Funktionen :
          (DEFUN PutToCol   (Table key element) ... )
          (DEFUN DelFromCol (Table key element) ... )
          (DEFUN ExistCol?  (Table key) ... )
          (DEFUN CreateCol  (Table key) ... )
          (DEFUN GetCol     (Table key) ... )
          (DEFUN DelCol     (Table key ) ... )
```

Modul : **AOHILF.LSP**

Beschreibung : Der Modul mit (z.T. internen) Hilfsfunktionen

```
Exportierte Funktionen :
          (DEFUN Schluesselliste (assocli) ... )
          (DEFUN Unik (L) ... )
          (DEFUN LDiff (L1 L2) ... )
          (DEFUN Remove (list ele) ... )
          (DEFUN ARemove (list key) ... )
          (DEFUN UpdProp (new key assoclist) ... )
          (DEFUN GetProp (key assoclist) ... )
          (DEFUN GenSym () ... )
          (DEFUN GenInstName () ... )
          (DEFUN GenMetName () ... )
          (DEFUN Symbol? (Arg) ... )
          (DEFUN ExprToString (Expr) ... )
```

2.4. Referenz

Es folgt nun die Beschreibung aller verfügbaren Funktionen. Zur genaueren Analyse sei auf deren Quelltext in den entsprechenden Files verwiesen. Für Funktionen, für die Effizienz besonders wichtig ist, wird unter dem kursiv-fett gedruckten Funktionsnamen eine kursiv aber nicht fett gedruckte Funktion aufgeführt. Deren Name beginnt mit '**S**' (für **s**hort), gefolgt von einem entsprechenden Buchstabenkürzel. In diesen 'short-versions' wird auf jegliche Fehlertests verzichtet.
Es sei hier nochmals darauf verwiesen, daß durch Speicherung aller Objektinformationen in Form von ASSOC-Listen als Wert von Lispsymbolen eventuell alte Informationen überschrieben werden. Es dürfen alle Symbolnamen nur einmal verwendet werden. Entsprechende Fehlertests beim Erzeugen von Objekten können nicht vorgenommen werden! Auf die Benutzung von Symbolen, die mit '$' beginnen, sollte verzichtet werden, da diese der Organisation des Objektsystems dienen.

2.4.1. Initialisierung des Objektsystems

Alle Symbole zur internen Verwendung in ALOS beginnen mit einem **$**Zeichen und sind max. 6 Zeichen lang. Entsprechende Symbolnamen dürfen durch Programmierer nicht verwendet werden.

NICHTS Definieren eines Symbols NICHTS, das mit 'NICHTS belegt ist. Es schafft die Möglichkeit Attribute zu erzeugen, diese aber nicht zu belegen (d.h. eben auch nicht mit NIL).

$SYM0 enthält den aktuellen Zählerstring zur Generierung von Symbolen (beginnen bei "$SAAAA" ... "$SXXXX")

$INST0 enthält den aktuellen Zählerstring zur Generierung von internen Instanzsymbolen (beginnen bei "$IAAAA" ... "$IXXXX")

$MET0 enthält den aktuellen Zählerstring zur Generierung von internen Methodensymbolen (beginnen bei "$MAAAA" ..."$MXXX")
$ITAB0 enthält die aktuelle Instanztabelle als ASSOC-Liste (initial mit ())

Das Laden des AutoLisp-Objektsystems erfolgt durch Ruf von
(LOAD "ALOS.LSP") ,
zweckmäßiger Weise in 'ACAD.LSP'. Hierbei erfolgt das Anlegen und Initialisieren der Systemsymbole.

(ResetALOS)
Rücksetzen des Objektsystems
Seiteneffekte : Setzen aller Systemsymbole auf ihre initialen Werte, alle Nutzerobjekte bleiben zwar erhalten, sind aber nicht mehr in das Objektsystem eingebunden.
Return : T
Beispiel: (ResetALOS)
> T

2.4.2 Erzeugen von Objekten, Slots und Klassenmonitoren

Erzeugen von Objekten

***(ErzeugeKlasse** name super)*
Dynamisches Erzeugen einer Klasse in einer Taxonomie. Wenn das Symbol *name* existiert, wird es überschrieben! Initial verfügen die Klassen lokal nur über die Standardslots.
Das Objekt *super* muß existieren. Das Argument *name* kann Symbol sein, dann werden die Objektinformationen in Form einer ASSOC-Liste in dem Symbol abgelegt. Wenn *name* ein String war, muß dieser sich in ein entsprechendes Symbol wandeln lassen, gleichzeitig wird der Namensslot entsprechend belegt.
Argumente : *name* - Symbol oder String
super - (ALOS-Objekt)
Seiteneffekte : - Setzen des Symbols *name* auf eine ASSOC-Liste mit den Basisin-
Basisinformationen des Objektes
- Verändern des SUB-Slots in *super*
Fehler : 'ERZEUGEKLASSE_NO_SUPER
super
Der symbolische Ausdruck *super* ist kein gültiges ALOS-Objekt vom Typ
Klasse.
'ERZEUGEKLASSE_WRONG_ARGUMENTTYP
super
Der symbolische Ausdruck *name* ist kein lesbarer String und kein Symbol.
Return : Klassenname
Beispiel: (ErzeugeKlasse 'Flaeche 'EXTERN)
> FLAECHE

***(ErzeugeStatischeKlasse** name super)*
siehe ***ErzeugeKlasse***
Im Gegensatz dazu verfügen die Klassen jedoch lokal über die Standardslots, sowie lokal über alle ererbten Slots !
Argumente : *name* - Symbol oder String
super - (ALOS-Objekt)
Seiteneffekte : - Setzen des Symbols *name* auf eine ASSOC-Liste mit den Basisin-
formationen des Objektes
- Verändern des SUB-Slots in *super*
Fehler : 'ERZEUGEKLASSE_NO_SUPER
super
Der symbolische Ausdruck *super* ist kein gültiges ALOS-Objekt vom Typ
Klasse.
'ERZEUGEKLASSE_WRONG_ARGUMENTTYP
super
Der symbolische Ausdruck *name* ist kein lesbarer String und kein Symbol.
Return : Klassenname
Beispiel: (ErzeugeStatischeKlasse 'Polygon_FL 'Flaeche)
> POLYGON_FL

***(ErzeugeInstanz** name super)*
Dynamisches Erzeugen einer Instanz von einer Klasse. Ist *name* vom Typ Symbol, wird kein Symbol generiert und als Namensstring der ererbte verwendet. Dieser kann später ggf. mit ***AendereName*** geändert werden. Wird als erstes Argument ein String angegeben, so wird ein internes Instanzsymbol vergeben (z.B. $IAABT) und dieses Symbol wird unter dem Schlüssel *name*(-string) in die Instanztabelle $ITAB0 eingetragen. (siehe auch ***ErzeugeStatischeInstanz***) Ist der String leer (""), so wird der ererbte Name verwendet. Initial verfügt die Instanz lokal nur über die Standardslots.
Argumente : *name* - String oder Symbol
super - Symbol (ALOS-Klasse)
Seiteneffekte : - Setzen des Objektsymboles auf eine ASSOC-Liste mit den Basisinformationen der Instanz.
- Verändern der Instanztabelle.
- Verändern des INST-Slots in *super*
Fehler : 'ERZEUGEINSTANZ_NO_CLASS
super
Der symbolische Ausdruck *super* ist kein gültiges ALOS-Objekt vom Typ Klasse
Return : Instanzsymbol
Beispiel: (ErzeugeInstanz "D3" 'Dreiecke)
> $IAABT

***(SEI** super)*
Dynamisches Erzeugen einer Instanz von einer Klasse. Es handelt sich hier um eine effiziente Kurzform, ohne jeden Fehlertest. Es wird ein internes Instanzsymbol vergeben (z.B. $IAABT) in dem die Objekt informationen abgelegt werden. Dieses Symbol wird unter dem Schlüssel des ererbten Namensstrings in die Instanztabelle $ITAB0 eingetragen. Der Name kann später ggf. mit **AendereName** geändert werden. Initial verfügt die Instanz lokal nur über die Standardslots. In der Kurzform feuert der 'BEIERZEUGEN - Monitor nicht ! Bei Bedarf kann dies aber jederzeit durch explizites Triggern erfolgen.
Argumente : *super* - Symbol (ALOS-Klasse)
Seiteneffekte : - Setzen des Objektsymbols auf eine ASSOC-Liste mit den Basisinformationen der Instanz.
- Verändern der Instanztabelle.
- Verändern des INST-Slots in *super*
Fehler : 'SEI_NO_SUPER
super
Der symbolische Ausdruck *super* ist kein gültiges ALOS-Objekt vom Typ Klasse
Return : Instanzsymbol
Beispiel: (SEI 'Dreiecke)
> $IAABU

***(ErzeugeStatischeInstanz** inst super)*
Statisches Erzeugen einer Instanz einer Klasse. Initial verfügt die Instanz lokal über die Standardslots sowie alle ererbten Slots. Es bestehen folgende Varianten gemäß des Argumentes *inst* :
inst ist NIL - Es wird ein Symbol generiert, im Namensslot steht der ererbte Name
inst ist Symbol - Es wird das gegebene Symbol verwendet, im Namensslot steht der ererbte Name
inst ist "" - Es wird ein Symbol generiert, im Namensslot steht der ererbte Name
inst ist String - Es wird ein Symbol generiert, im Namensslot steht der gegebene Name
Das Objektsymbol wird unter dem Schlüssel des Namensslots in die Instanztabelle $ITAB0 eingetragen.
Argumente : *inst* - Symbol oder Symbol
super - Symbol (ALOS-Klasse)
Seiteneffekte : - Setzen des Symbols *inst* auf eine ASSOC-Liste mit den Basisinformationen der Instanz
- Verändern der Instanztabelle
- Verändern des INST-Slots in *super*

Fehler : 'ERZEUGESTATISCHINSTANZ_NO_CLASS
super
Der symbolische Ausdruck *super* ist kein gültiges ALOS-Objekt vom Typ Klasse
Return : Instanzsymbol
Beispiel: (ErzeugeInstanz 'D1 'Dreiecke)
> D1

***(KopiereInstanz** inst)*
(SKI inst)
Dynamisches Erzeugen einer Kopie einer Instanz. Es wird ein internes Instanzsymbol vergeben (z.B. $IAABT) in dem die Objektinformationen von *inst* abgelegt werden. Dieses Symbol wird unter dem Schlüssel des gleichen Namensstrings wie *inst* in die Instanztabelle $ITAB0 eingetragen. Der Name kann später ggf. mit **AendereName** geändert werden. Der 'BEIERZEUGE Monitor der Superklasse von *inst* feuert nur bei der Langform.
Argumente : *inst* - Symbol (ALOS-Instanz)
Seiteneffekte : - Setzen des generierten Objektsymbols auf eine ASSOC-Liste mit den Basisinformationen der Instanz.
- Verändern der Instanztabelle.
- Verändern des INST-Slots im Superobjekt von *inst*
Fehler : 'KOPIEREINSTANZ_NO_INSTANCE
inst
Der symbolische Ausdruck *inst* ist kein gültiges ALOS-Objekt vom Typ Instanz

Return : Instanzsymbol
Beispiel: (KopiereInstanz 'Dreieck13)
> $IAABU

Erzeugen von Slots

***(ErzeugeAttribut** class attr init)*
Erzeugen eines Attributes an einer Klasse. Wenn das Attribut existiert, wird es überschrieben! (zum Ändern nicht effizient!)
Argumente : *class* - Typ Symbol (ALOS-Objekt)
attr - Typ Symbol (Attributname)
init - beliebiger Typ (Attributwert)
Seiteneffekte : verändert die ASSOC-Liste in *class*
Fehler : 'ERZEUGEATTRIBUT_NO_OBJEKT
attr init
Der symbolische Ausdruck *class* ist kein gültiges ALOSKlasse
'ERZEUGEATTRIBUT_ONLY_CLASSES
attr init
class ist nicht vom Typ Klasse
Return : Attributname
Beispiel: (ErzeugeAttribut 'DREIECKE 'Farbe 'grün)
> FARBE

***(ErzeugeMethode** class met paramlist metkoerperlist)*
Erzeugen einer Methode an einer Klasse mit Angabe des Methodenkör pers als LAMBDA-Liste. Die Methode selbst wird unter einem internen Funktionssymbol (z.B. $MAAXY) abgelegt. Der Methodenslot enthält einen ungebundenen LAMBDA-Ausdruck mit dem entsprechenden Aufruf. Es wird durch ***ErzeugeMethode*** sowohl im LAMBDA-Ausdruck als auch in der internen Funktion das Argument **SELF** der Argumentliste hinzugefügt. Beim Triggern der Methode wird es an das Objekt gebunden, das die Nachricht erhielt (auch wenn der Methodentext ererbt wurde).

Argumente : *class* - Typ Symbol (ALOS-Objekt)
met - Typ Symbol (Methodenname)
paramlist - Typ Liste (Liste der Parameter, auch lokale Symbole)
metkoerperlist - Typ Liste (LAMBDA-Ausdruck der Methode)
Seiteneffekte : Ändern der ASSOC-Liste in *class*, Erzeugen einer Funktion in einem internen Symbol.
Fehler : 'ERZEUGEMETHODE_NO_OBJEKT
met paramlist
Der symbolische Ausdruck *class* ist keine gültige ALOS-Klasse.
Return : Methodenname
Beispiel: (ErzeugeMethode 'Flaeche 'umfang_ber '(arg1 / locarg1)
'((prompt " Eine ausführbare Methode. ")
(* (SendeNachricht Self flaeche '()) arg1)
))
> UMFANG_BER

Erzeugen von Monitoren

***(ErzeugeKlassenMonitor** class monitor locsymlist metkoerperlist)*
Erzeugen einer Methode an einer Klasse mit dem Namen *monitor* mit Angabe des Methodenkörpers als Liste. Folgende Monitore werden unterstützt:
BeiErzeugen - feuert beim Erzeugen einer Instanz nach deren physischer Erzeugung
BeiLoeschen - feuert beim Löschen einer Instanz vor deren physischer Löschung
Die folgenden Monitore sind bei der Verwendung des **ALAS**- Systems verfügbar :
BeiEinbau - feuert beim Einbau einer Instanz in ein Komplexobjekt nach dem physischen Einbau.(Module **ALAS**)
BeiAusbau - feuert beim Ausbau einer Instanz aus ein Komplexobjekt vor dem physischen Ausbau.(Modul **ALAS**)
Die Methode selbst wird unter einem internen Funktionssymbol (z.B. $MAAXY) abgelegt, der Methodenslot enthält einen ungebundenen LAMBDA - Ausdruck mit dem entsprechenden Aufruf. Neben den lokalen Symbolen in *locsymlist* ist der Parameter **SELF** immer verfügbar (Instanz, die erzeugt, gelöscht, ein- oder ausgebaut wird).
Argumente : *class* - Typ Symbol (ALOS-Klasse)
monitor - Typ Symbol (Monitorname)
locsymlist - Typ Liste von Symbolen (Liste der lokalen Symbole)
metkoerperlist - Typ Liste (LAMBDA-Ausdruck der Methode)
Seiteneffekte : Ändern der ASSOC-Liste in *class*, Erzeugen einer Funktion in einem internen Symbol.
Fehler : 'ERZEUGEKLASSENMONITOR_NO_OBJEKT
monitor
Der symbolische Ausdruck *class* ist keine gültige ALOS-Klasse.
'ERZEUGEKLASSENMONITOR_NO_CLASSOBJEKT
monitor
Der symbolische Ausdruck *class* ist nicht vom Typ Klasse.
'ERZEUGEKLASSENMONITOR_NO_MONITOR
monitor
Es existiert kein Monitor.
Return : Monitorname
Beispiel: (ErzeugeKlassenMonitor 'Flaeche 'BeiErzeugen '(locarg1 la2)
'((PRINT "Erzeugt wurde : ")(PRINT SELF)
(PRINT "Mit Namen : ") (PRINT (SN SELF 'NAME '())
))
> BEIERZEUGE

2.4.3. Löschen von Objekten und Slots

Löschen von Objekten

***(LoescheInstanz** inst*)
(SLOI inst)
Löschen einer Instanz (nur kurze Typprüfung). Monitor 'BeiLoeschen' feuert wenn vorhanden.
Argumente : *inst* - Typ Symbol (ALOS-instanz)
Seiteneffekte : das Instanzsymbol wird auf NIL gesetzt
Fehler : 'LOESCHEINSTANZ_NO_INSTANZ
Der symbolische Ausdruck in *inst* ist keine gültige ALOS-Instanz
Return : NIL
Beispiel : (LoescheInstanz 'D3)
> NIL

***(LoescheInstanzen** class)*
Loeschen aller Instanzen einer Klasse (nur kurze Typprüfung), auch der indirekten Instanzen, d.h. der Instanzen von Subklassen!
Monitor 'BeiLoeschen' feuert bei jeder Instanz, wo er verfügbar ist.
Argumente : *class* - Typ Symbol (ALOSKlasse)
Seiteneffekte : Die Symbole aller Unterobjekte von *class* die Instanzen sind,
werden auf NIL gesetzt.
Fehler : 'LOESCHEINSTANZEN_NO_CLASS
Der symbolische Ausdruck *class* ist keine gültige ALOS-Klasse
Return : NIL
Beispiel : (LoescheInstanzen 'Vierecke)
> NIL

***(LokLoescheInstanzen** class)*
Löschen aller direkten Instanzen einer Klasse (nur kurze Typprüfung) Monitor 'BeiLoeschen' feuert bei jeder Instanz, wenn er bei *class* vorhanden ist.
Argumente : *class* - Typ Symbol (ALOS-Klasse)
Seiteneffekte : Die Symbole aller direkten Unterobjekte von *class* die Instanzen sind, werden auf NIL gesetzt.
Fehler : 'LOKLOESCHEINSTANZEN_NO CLASS
Der symbolische Ausdruck *class* ist keine gültige ALOS-Klasse
Return : NIL
Beispiel : (LokLoescheInstanzen 'Dreiecke)
> NIL

***(LoescheKlasse** class)*
Loeschen einer Klasse (nur kurze Typprüfung). Alle Subklassen und direkten Instanzen werden Subobjekte der alten Superklassen, d.h. es wird echt nur *class* gelöscht.
Da keine Instanzen gelöscht werden, feuert kein Monitor.
Argumente : *class* - Typ Symbol (ALOS-Klasse)
Seiteneffekte : Das Symbol *class* wird auf NIL gesetzt.
Fehler : 'LOESCHEKLASSE_NO_CLASS
Der symbolische Ausdruck *class* ist keine gültige ALOS-Klasse
Return : NIL
Beispiel : (LoescheKlasse 'Vierecke)
> NIL

***(LoescheKlasseGanz** class)*
Löschen einer Klasse (nur kurze Typprüfung) einschließlich aller ihrer direkten und indirekten Subklassen. Alle direkten und indirekten Instanzen werden Subobjekte der alten Superklassen.
Da keine Instanzen gelöscht werden feuert kein Monitor.

Argumente : *class* - Typ Symbol (ALOS-Klasse)
Seiteneffekte: Das Symbol *class* sowie die Symbole aller Subklassen werden auf NIL gesetzt.
Fehler : 'LOESCHEKLASSEGANZ_NO_CLASS
Der symbolische Ausdruck *class* ist keine gültige ALOS-Klasse
Return : NIL
Beispiel : (LoescheKlasse 'Wurzel)
> NIL

Löschen von Slots

***(LokLoeschenSlot** obj slot)*
(SLLOS obj slot)
Lokales löschen eines Slots. Eignet sich auch zum 'wieder global machen' eines Slots, d.h. der Wert wird durch Erbschaft verfügbar. Wird allerdings versucht die Slotdefinition zu löschen, erfolgt lediglich das Belegen des Slots mit 'NICHTS. In der Kurzversion wird jedoch kommentarlos gelöscht ! Das kann zur Folge haben, daß der *slot* lokal in einer Subklasse verfügbar bleibt!
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
slot - Typ Symbol - Slotname
Seiteneffekte: Der Slot wird aus der ASSOC-Liste des Objektsymbols gelöscht
Fehler : 'LOKLOESCHESLOT_NO_OBJEKT
slot
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
'LOKLOESCHESLOT_NO_SLOT
slot
Der symbolische Ausdruck *slot* ist kein gültiger Slot
'LOKLOESCHESLOT_STANDARD_SLOT
slot
Es wird versucht einen Standardslot zu löschen. (z.B. 'TYP)
Return : NIL
Beispiel: (LokLoescheSlot 'Dreiecke 'Form)
> NIL

***(LoescheSlot** obj slot)*
(SLOS obj slot)
Löschen eines Slots bei *obj* und all seinen Subobjekten. Eignet sich auch zum 'wieder global machen' eines Slots für einen gesamten Teilbaum, d.h. im gesamten Teilbaum wird der Slotwert über Erbschaft bezogen. Soll der Slot völlig aus dem Objektsystem entfernt werden, ist wie folgt zu verfahren :
(LoescheSlot (WoSlotDefinition obj slot) slot)
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
slot - Typ Symbol - Slotname
Seiteneffekte : Der Slot wird aus der ASSOC-Liste des Objektsymbols gelöscht
Fehler : 'LOESCHESLOT_NO_OBJEKT
slot
Der symbolische Ausdruck *obj* ist kein gültiges ALOSObjekt.
'LOESCHESLOT_NO_SLOT
slot
Der symbolische Ausdruck *slot* ist kein gültiger Slot.
Return : NIL
Beispiel : (LoescheSlot 'Dreiecke 'Farbe)
> NIL

2.4.4. Versenden von Nachrichten

***(SendeNachricht** obj msg paramlist)*
(SN obj msg paramlist)
Senden einer Nachricht an ein Objekt, dabei wird wie folgt vorgegangen:
- holen des Inhalts des Slots *msg* von *obj*, ggf. über Erbschaft
- ist der Wert kein LAMBDA-Ausdruck wird er zurück gegeben, sonst wird die Parameterliste um den Parameter *obj* verlängert (als formales Argument SELF immer in der Methode verfügbar) und der LAMBDA-Ausdruck wird auf die Parameterliste angewendet.

Empfänger von Nachrichten können Instanzen und Klassen sein!
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
msg - Typ Symbol (Slotname)
parameterlist- Typ List (Liste der Parameter einer Methode)
Seiteneffekte : keine eigenen
Fehler : 'SENDENACHRICHT_NO_OBJEKT
msg paramlist
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt.
'SENDENACHRICHR_NO_SLOT
msg paramlist
Der symbolische Ausdruck *msg* ist kein gültiger Slot
Return : Bei Nachrichten an Attribute : Attributwert
Bei Nachrichten an Methoden : Ergebnis der Auswertung
Beispiel: (ErzeugeMethode 'Dreicke 'Test_Fkt '(/ locparam)
'((SETQ locparam SELF)
(PRINC "Beispielnachricht an : ")(PRINT locparam)
))
>TEST_FKT
(SendeNachricht 'Dreieck23 'Test_Fkt '())
>Beispielnachricht an : DREIECK23
(SN 'DREIECKE 'FARBE '())
>HELLROT
(ErzeugeMethode 'Dreicke 'Umfang '(faktor)
'((* faktor (+ (SN SELF 'S1 '())
(SN SELF 'S2 '())
(SN SELF 'S3 '())
))))
>UMFANG
(SN 'Dreieck25 'UMFANG (LIST 1000))
>12000

2.4.5. Ändern von Slotbelegungen und Klassenzugehörigkeiten

Ändern der Klassenzugehörigkeit

***(AendereSuper** obj newsuperclass)*
Umhängen eines Objektes (bei Klassen ganzer Teilbaum) in der Taxonomie. Für *obj* und alle seine Subobjekte werden alle Slots gelöscht, die nicht im Teilbaum von *obj* definiert sind und in der neuen Super klasse *newsuperclass* nicht verfügbar sind. Der 'BeiLoeschen' Monitor feuert nicht.
Argumente : o*bj* - Typ Symbol (ALOS-Objekt)
newsuperclass - Typ Symbol (ALOS-Objekt)
Seiteneffekte : Die ASSOC-Liste von *obj*, *newsuperclass* und der alten Superklasse sowie aller Objekte, bei denen lokal Slots gelöscht werden, werden verändert.

Fehler : 'AENDERESUPER_NO_OBJEKT
newsuperclass
Der symbolische Ausdruck *obj* oder *newsuperclass* ist kein gültiges ALOS-Objekt.
'AENDERESUPER_NO_NEWSUPEROBJEKT
newsuperclass
Der symbolische Ausdruck *newsuperclass* ist kein ALOS-Objekt vom Typ KLasse.
Return : Neue Superklasse - Typ Symbol
Beispiel: (AendereSuper '$IAAAG 'Flaeche)
> FLAECHE
(AendereSuper 'Dreiecke 'Figuren)
> FIGUREN

Ändern von Slots

***(LokAendereAttribut** obj attr new)*
(SLAe obj attr new)
Lokales Ändern eines Attributwertes. Wenn das Attribut lokal nicht existiert, wird er lokal angelegt und mit *new* initialisiert. Ist der Slot nicht verfügbar, erfolgt eine Fehlermeldung.
Achtung : Nicht zur Änderung des 'NAME'-Slots verwenden!
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
attr - Typ Symbol (Attributname)
new - neuer Attributwert
Seiteneffekte : verändert die ASSOC-Liste in *obj*
Fehler : 'LOKAENDEREATTRIBUT_NO_OBJEKT
attr new
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt.
'LOKAENDEREATTRIBUT_NO_SLOT
attr new
Der Slot *attr* ist nicht verfügbar.
'LOKAENDEREATTRIBUT_NOT_FOR_NAME
attr new
Der Slot *attr* ist 'NAME'.
Retur : neuer Attributwert
Beispiel: (LokAendereAttribut 'd1 'Seite_da 50)
> 50

***(GlobalAendereAttribut** obj attr new)*
(SAe obj attr new)
Ändern eines Attributwertes (auch über Erbschaft), d.h. die Änderung erfolgt dort, von wo die Information bezogen wird. Wenn *attr* nicht existiert wird eine Fehlermeldung ausgegeben.
Achtung: Die Nutzung dieser Funktion kann 'unschöne' Seiteneffekte haben! (Änderung beeinflußt durch Erbschaft andere Objekte)
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
attr - Typ Symbol (Attributname)
new - neuer Attributwert
Seiteneffekte : verändert die ASSOC-Liste in dem Objekt, von wo der Attributwert bezogen wird, d.h. es tritt eine Werteänderung überall dort ein, wo der Attributwert von *attr* durch Erbschaft bezogen wurde!
Fehler : 'GLOBALAENDEREATTRIBUT_NO_OBJEKT
attr new
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt.
'GLOBALAENDEREATTRIBUT_NO_ATTRIBUT
attr new
Das Attribut *attr* ist kein gültiges Attribut im Objekt *obj* Return : neuer Attributwert
Beispiel: (GlobalAendereAttribut 'Dreieck_23 'Seite_anzahl 3.5)
> 3.5

***(SetO** obj attr new)*
Ändern eines Attributwertes. Es handelt sich hierbei um die Basisfunktion aller anderen *Aendere*-Funktionen und ist als solche eigentlich intern. Geändert wird lokal und nur wenn das Attribut lokal verfügbar ist, sonst gibt es keine Aktion.
Achtung : *SetO* ändert kommentarlos jeden Slot.
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
attr - Typ Symbol (Attributname)
new - neuer Attributwert
Seiteneffekte : verändert die ASSOC-Liste in *obj*
Fehler : -
Return : neuer Attributwert
Beispiel: (SetO 'Dreieck 'NAME "ist Dreieck")
> "ist Dreieck" ; ← kein Ändern der Instanzliste !!

***(AendereName** obj newnamestring)*
(SAN obj newnamestring)
Ändern des 'NAME'-Slots eines Objektes. Ist das *obj* vom Typ Instanz, wird die Instanztabelle $ITAB0 aktualisiert.
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
newnamestring - neuer Namensstring
Seiteneffekte : verändert die ASSOC-Liste in *obj* und ggf. der Instanztabelle
Fehler : 'AENDERENAME_NO_OBJEKT
newnamestring
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt.
Return : neuer Namensstring
Beispiel: (AendereName '$IAABF "Dachdreieck")
> "Dachdreieck"

2.4.6 Prädikative Funktionen

***(ALOSObjekt?** obj)*
Prüft, ob ein symbolischer Ausdruck ein gültiges ALOS-Objekt ist. Wenn nicht erfolgt eine Fehlerausschrift.
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : ev. alph.Bildschirmausgabe
Fehler : 'ALOSOBJEKT?_NO_SYMBOL
Der symbolische Ausdruck *obj* ist kein gültiges ALOSObjekt.
'ALOSOBJEKT?_NIL_IN_SYMBOL
Der symbolische Ausdruck *obj* ist mit NIL belegt.
'ALOSOBJEKT?_NO_LIST_IN_SYMBOL
Der symbolische Ausdruck *obj* ist nicht mit einer Liste belegt.
'ALOSOBJEKT?_NO_ASSOCLIST_IN_SYMBOL
Der symbolische Ausdruck ist nicht mit einer ASSOC-Liste belegt.
'ALOSOBJEKT?_UNCOMPLETE_OBJEKTDATA
Der symbolische Ausdruck ist keine ASSOC-Objektliste.
Return : T/NIL (bei NIL Fehlerauschrift und Abbruchmöglichkeit)

Beispiel: (ALOSObjekt? 'Dreiecke)
> T
(ALOSObjekt? 'quatsch)
> Fehler in Funktion/Methode: ALOSOBJEKT?_NIL_IN_SYMBOL
Fehler bei Objekt : QUATSCH
--- Parameterliste ---
--- Abbruch mit ^C ---
NIL

***(ALOSObj?** obj)*
Prüft, ob ein symbolischer Ausdruck ein gültiges ALOS-Objekt ist (analog ALOSObjekt?, aber ohne Seiteneffekte).
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekt : keine
Fehler : Keine
Return : T/NIL
Beispiel: (ALOSObj? 'D1)
> T
Beispiel: (ALOSObj? 'quatsch)
> NIL

***(OTyp** obj)*
Gibt den ALOS-Objekttyp eines übergebenen Symbols zurück oder NIL. Eignet sich zur verkürzten Prüfung ob ALOS-Objekt vorliegt. Achtung! Aus Effizienzgründen erfolgt nur eine oberflächliche Typprüfung. Zur genauen Prüfung Funktion ALOSObj? verwenden !
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : keine
Return : 'KLASSE, 'INSTANZ oder NIL
Beispiel: (OTyp 'Flaeche)
> KLASSE
(OTyp 'D1)
> INSTANZ
(OTyp 'quatsch)
> NIL

***(Slot?** obj slot)*
(SS? obj slot)
Ist das Slot *slot* im Objekt *obj* verfügbar ?
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
slot - Typ Symbol (Slotname)
Seiteneffekte : keine
Fehler : 'SLOT?_NO_OBJEKT
slot
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : ASSOC-Listenelement (Key Value) oder NIL
Beispiel: (Slot? 'Dreiecke 'Eck_Anzahl)
> (ECK_ANZAHL 3)
(Slot? 'd1 'Seite_da)
> (SEITE_DA 30)
(Slot? 'd1 'noslot)
> NIL

***(LokSlot?** obj slot)*
(SLS? obj slot)
Ist das Slot *slot* im Objekt *obj* lokal verfügbar ?
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
slot - Typ Symbol (Slotname)
Seiteneffekte : keine
Fehler : 'LOKSLOT?NO_OBJEKT
slot
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt.
Return : ASSOC-Listenelement (Key Value) oder NIL
Beispiel: (LokSlot? 'Dreiecke 'Seite_da)
> (SEITE_DA 10)
(LokSlot? 'd1 'Seite_da)
> (SEITE_DA 30)
(LokSlot? 'd1 'noslot)
> NIL

***(Methode?** obj met)*
Ist die Methode *met* im Objekt *obj* verfügbar ?
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
met - Typ Symbol (Methodenname)
Seiteneffekte : keine
Fehler : 'METHODE?_NO_OBJEKT
met
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt.
Return : ASSOC-Listenelement (Key Value) oder NIL
Beispiel: (Methode? 'Flaeche 'Umfang_ber)
> (UMFANG_BER (LAMBDA (SELF) ($MAACG SELF)))

***(LokMethode?** obj met)*
(SLM? obj met)
Ist die Methode *met* im Objekt *obj* lokal verfügbar?
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
met - Typ Symbol (Methodenname)
Seiteneffekte : keine
Fehler : 'LOKMETHODE?_NO_OBJEKT
met
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : ASSOC-Listenelement (Key Value) oder NIL
Beispiel: (LokMethode? 'Dreiecke 'Umfang_ber)
> (UMFANG_BER (LAMBDA (SELF) ($MABCD SELF)))

2.4.7. Analyse der Struktur von Objekten

***(LokMethoden** obj)*
(SLM obj)
Erzeugen der Liste der lokalen Methoden des Objektes *obj*
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'LOKMETHODEN_NO_OBJEKT
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : Liste mit Methodennamen
Beispiel: (LokMethoden 'Ellipse)
> (FLAECHEN_BER VISU_DICH)

***(Methoden** obj)*
(SM obj)
Erzeugen der Liste aller verfügbarer Methoden
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'METHODEN_NO_OBJEKT
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : Liste mit Methodennamen
Beispiel: (Methoden 'Ellipse)
> (VISU_DICH FLAECHEN_BER UMFANG_BER)

***(LokAttribute** obj)*
Erzeugen der Liste der lokalen Attribute
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'LOKATTRIBUTE_NO_OBJEKT
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : Liste mit Attributnamen
Beispiel: (LokAttribute 'D1)
> (SUPER TYP SEITE_DA SEITE_DB SEITE_DC)

***(Attribute** obj)*
Erzeugen der Liste aller verfügbarer Attribute
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'ATTRIBUTE_NO_OBJEKT
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : Liste mit Attributnamen
Beispiel: (Attribute 'D1)
> (SEITE_DC SEITE_DB SEITE_DA ECK_ANZAHL FORM
GESTALT TYP SUPER)

***(LokSlots** obj)*
(SLS obj)
Erzeugen der Liste der lokalen Slots
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'LOKSLOTS_NO_OBJEKT
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : Liste mit Slotnamen
Beispiel : (LokSlots 'D1)
> (SEITE_DC SEITE_DB SEITE_DA TYP SUPER)

***(Slots** obj)*
(SS obj)
Erzeugen der Liste aller verfügbarer Slots
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'SLOTS_NO_OBJEKT
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : Liste mit Slotnamen
Beispiel: (Slots 'D1)
> (ECK_ANZAHL SEITE_DC SEITE_DB SEITE_DA SUPER TYP
GESTALT FORM UMFANG_BER FLAECHEN_BER VISU_DICH)

***(WoSlot** obj slot)*
(SWS obj slot)
Ermitteln des nächsten höheren Objekts, bei dem ein Slotwert geholt werden kann.
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
slot - Typ Symbol (Slotname)
Seiteneffekte : keine
Fehler : 'WOSLOT_NO_OBJEKT
slot
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : Symbol (ALOS- Objektname) oder NIL wenn nicht verfügbar
Beispiel: (WoSlot 'D1 'Form)
> (DREIECKE)

***(WoSlotDefinition** obj slot)*
Gibt das Superobjekt zurück, bei dem der Slot erstmalig definiert ist
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
slot - Typ Symbol - Slotname
Seiteneffekte : keine
Fehler : 'WOSLOTDEFINITION_NO_OBJEKT
slot
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt.
Return : Objektsymbol oder NIL wenn nicht verfügbar.
Beispiel: (WoSlotDefinition 'D1 'PART-OF)
> EXTERN

2.4.8. Analyse der Struktur der Taxonomie

***(Vorgaenger** obj)*
(SV obj)
Erzeugen der Vorgängerliste
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'VORGAENGER_NO_OBJEKT
Der symbolische Ausdruck *obj ist* kein gültiges ALOS-Objekt
Return : Liste mit Symbolen (ALOS-Objekte)
Beispiel: (Vorgaenger 'd1)
> (DREIECKE POLYGON_FL FLAECHE)

***(Nachfolger** class)*
***(SNF** class)*
Erzeugen der linearen Liste aller Nachfolgeobjekte (Klassen und Instanzen). Bei Instanzen wird NIL zurück gegeben.
(nur mit Kurzprüfung!)
Argumente : *class* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'NACHFOLGER_NO_OBJEKT
Der symbolische Ausdruck *class* ist kein gültiges ALOS-Objekt.
Return : Liste mit Symbolen (ALOS-Objekte)
Beispiel: (Nachfolger 'Figuren)
> (VIERECKE $IAABX V12 $IABCF DREIECKE $IABGF' D1)

***(NachfolgerKlassen** class)*
Erzeugen der linearen Liste aller Subklassen
Argumente : *class* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'NACHFOLGERKLASSEN_NO_OBJEKT
Der symbolische Ausdruck *class* ist kein gültiges ALOS-Objekt
Return : Liste mit Symbolen (ALOS-Objekte)
Beispiel: (NachfolgerKlassen 'Polygon_FL)
> (N_ECK VIERECKE RECHTECK DREIECKE)

***(LokNachfolgerKlassen** class)*
(SLNKl class)
Erzeugen der Liste der direkten Subklassen
Argumente : *class* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'LOKNACHFOLGERKLASSEN_NO_OBJEKT
Der symbolische Ausdruck *class* ist kein gültiges ALOS-Objekt
Return : Liste mit Symbolen (ALOS-Objekte)
Beispiel: (LokNachfolgerKlassen 'Polygon_Fl)
> (DREIECKE VIERECKE N_ECK)

***(Instanzen** class)*
(SI class)
Erzeugen der Liste aller Instanzen (auch der indirekten)
Argumente : *class* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'INSTANZEN_NO_OBJEKT
Der symbolische Ausdruck *class* ist kein gültiges ALOS-Objekt oder nicht vom Typ Klasse
Return : Liste mit Symbolen (ALOS-Objekte)
Beispiel: (Instanzen 'Vierecke)
> (V1 R1)

***(LokInstanzen** class)*
(SLI class)
Erzeugen der Liste der direkten Instanzen
Argumente : *class* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'LOKINSTANZEN_NO_OBJEKT
Der symbolische Ausdruck *class* ist kein gültiges ALOS-Objekt oder nicht von Typ Klasse
Return : Liste mit Symbolen (ALOS-Objekte)
Beispiel: (LokInstanzen 'Vierecke)
> (V1)

***(OberKlasse** obj1 obj2)*
Gibt die Klasse zurück, deren Subklassen *obj1* und *obj2* direkt oder indirekt sind.
Argumente : *obj1, obj2* - Typ Symbol (ALOS-Klasse)
Seiteneffekte : keine
Fehler : 'OBERKLASSE_NO_OBJEKT
obj1
'OBERKLASSE_NO_OBJEKT
obj2
Der symbolische Ausdruck *obj1, obj2* sind keine gültigen ALOS-Objekt oder nicht von Typ Klasse
Return : ALOS Klassensymbol
Beispiel: (Oberklasse 'Vierecke 'INTERN)
> 'ROOT

***(HoleName** obj)*
(SHN obj)
Holen des Namensstrings eines Objektes
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : keine
Fehler : 'HOLEINSTANZNAME_NO_OBJEKT
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : Namensstring
Beispiel: (HoleNamen '$IABCD)
> "Dreieck38"
(SHN 'EXTERN)
> "Externobjekt"

***(HoleInstanzSymbol** instname)*
(SHSYM instname)
Holen der Liste (!) von Objektsymbolen, die den Namen *instname* tragen aus der Instanztabelle $ITAB0
Argumente : *instname* - Typ String
Seiteneffekte : keine
Fehler : -
Return : Liste von Objektsymbolen
Beispiel: (HoleInstanzSymbol "Dreieck")
> ($IABCD $IASDR D124)

2.4.9. Zugriff auf Objektslots

Diese Zugriffsfunktionen dienen eigentlich der internen Organisation des Nachrichtenaustausches. Sie sind aber für den Zugriff auf Attribute wesentlich effizienter und werden deshalb hier beschrieben, obwohl sie eigentlich das Prinzip der Datenkapselung verletzen.

***(Hole** obj slot)*
(SH obj slot)
Holen des Slotwertes des Slots *slot* im Objekt *obj* (auch über Erbschaft)
Achtung : Ist reine Zugriffsfunktion !
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
slot - Typ Symbol (Slotname)
Seiteneffekte : Keine
Fehler : 'HOLE_ERR_NO_OBJEKT
slot
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
'HOLE_ERR_NO_SLOT
slot
Der symbolische Ausdruck *slot* ist kein gültiges Slot
Return : Slotwert (unausgewerteter S-Ausdruck)
Beispiel: (Hole 'D1 'UMFANG_BER)
> (LAMBDA (SELF) ($MAALX SELF))

***(LokHole** obj slot)*
(SLH obj slot)
Holen des Slotwertes des Slots *slot* im Objekt *obj* (nur lokal).
SLH ist die schnellste Zugriffsfunktion auf lokalen Objektdaten.
Achtung : Ist reine Zugriffsfunktion !
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
slot - Typ Symbol (Slotname)
Seiteneffekte : Keine
Fehler : 'LOKHOLE_NO_OBJEKT
slot
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
'LOKHOLE_NO_SLOT
slot

Der symbolische Ausdruck *slot* ist kein gültiges lokaler Slot
Return : Slotwert (unausgewerteter S-Ausdruck)
Beipiel: (LokHole 'D1 'Seite_da)
> 30

2.4.10. Lade und Sichern

Diese Funktionen tragen vorläufigen Charakter. Alle entstehenden Dateien tragen die Erweiterung '.LOB', diese Erweiterung braucht bei den Dateinamen nicht angegeben werden. Wenn eine Datei gleichen Namens existiert, wird diese kommentarlos überschrieben. Für die hier angeführten Funktionen wird in folgenden Versionen keine weiterentwicklung garantiert !
Die Dateien sind reine ASCII-Dateien und enthalten die Objektbeschreibung für jedes Objekt in folgender Form (jeweils eine Zeile):
(objektname objekt_daten_assoc_liste)

***(SichereALOS** outdatei)*
Sichern des gesamten aktuellen Objektsystems ab ROOT in eine Datei.
Argumente : *outdatei* - Typ String - (Path+Dateiname)
Seiteneffekte : Es wird Datei *outdatei* geschrieben
Fehler : 'SICHEREOBJEKTLISTE_CANT_OPEN_FILE
Die Datei *outdatei* läßt sich nicht öffnen.
'SICHEREOBJEKTLISTE_NO_OBJEKT
outdatei
Kein gültiges ALOSObjekt vorhanden.
Return : T / NIL
Beispiel: (SichereALOS "A:/Dreiecke")
> T

***(LadeALOS** indatei)*
Laden der Objekte aus *indatei*. Die Datei muß die Erweiterung ".LOB" tragen. Das vor dem Aufruf existierende Objektsystem wird über schrieben, alte Objekte (außer ROOT, EXTERN, INTERN) existieren noch (sofern *indatei* keine gleichnamigen Objektsymbole enthält). Sie sind aber nicht mehr mit dem Objektsystem verbunden.
Argumente : *indatei* - Typ String - (Dateiname)

Seiteneffekt : Es wird Datei *indatei* gelesen und die beschriebenen Objekte werden mit dem zugehörigen symbolischen Ausdruck belegt.
Fehler : 'LADEOBJEKTFILE_CANT_OPEN_FILE
Die Datei *indatei* läßt sich nicht öffnen
Return: T
Beispiel: (LadeALOS "d:/Dreiecke")
> T

2.4.11. Alphanumerische Anzeigefunktionen

Die alphanumerischen Anzeigefunktionen sind lediglich ein Hilfsmittel in der Entwicklungsphase mit dem Objektsystem.

***(ZeigeObjektGanz** obj)*
(SZOG pbj)
Ausgabe eines Objektes mit allen Informationen (auch ererbten)
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : alph. Bildschirmausgabe
Fehler : 'ZEIGEOBJEKTGANZ_NO_OBJEKT
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : *obj* - Objektsymbol
Beispiel: (ZeigeObjektGanz 'K1)

```
> ================ K1 ==================
TYP             : INSTANZ
SuperKlasse : KREIS
Attribut    : K_MITTELPUNKT :(10.0 10.0)
Attribut    : K_RADIUS : 40
Attribut    : GESTALT  :PLAN
Attribut    : FORM  : KONVEX
Methode  : UMFANG_BER
(LAMBDA (SELF) ($MAAXY SELF))
Methode  : FLAECHEN_BER
(LAMBDA (SELF) ($MAAXZ SELF))
Methode  : VISU_DICH
 (LAMBDA (SELF) ($MAAYA))
--- ENDE OBJEKT ---- TASTE DRUECKEN ____
K1
```

***(ZeigeObjekt** obj)*
(SZO obj)
Ausgabe eines Objektes nur mit lokalen Slots
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : alph. Bildschirmausgabe
Fehler : 'ZEIGEOBJET_NO_OBJEKT
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : *obj* - Objektsymbol

Beispiel: (ZeigeObjekt 'K1)

```
> ================ K1 ==================
TYP                 : INSTANZ
SuperKlasse    : KREIS
Attribut  : K_MITTELPUNKT :(10.0 10.0)
Attribut  : K_RADIUS : 40
--- ENDE OBJEKT ---- TASTE DRUECKEN ____
K1
```

***(ZeigeAttributeGanz** obj)*
(SZAG obj)
Ausgabe eines Objektes mit allen Attributen (auch ererbten)
Argumente : obj - Typ Symbol (ALOS-Objekt)
Seiteneffekte : alph. Bildschirmausgabe
Fehler : 'ZEIGEATTRIBUTGANZ_NO_OBJEKT
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : *obj* - Objektsymbol

Beispiel: (ZeigeAttributeGanz 'K1)
> ========== K1 ==========
Typ : INSTANZ
SuperKlasse : KREIS
Attribut : K_MITTELPUNKT :(10.0 10.0)
Attribut : K_RADIUS : 40
Attribut : GESTALT :PLAN
Attribut : FORM : KONVEX
--- ENDE OBJEKT ---- TASTE DRUECKEN -----
K1

***(ZeigeAttribute** obj)*
(SZA obj)
Ausgabe eines Objektes nur mit lokalen Attributen
Argumente : *obj* - Typ Symbol (ALOS-Objekt)
Seiteneffekte : alph. Bildschirmausgabe
Fehler : 'ZEIGEATTRIBUTE_NO_OBJEKT
Der symbolische Ausdruck *obj* ist kein gültiges ALOS-Objekt
Return : *obj* - Objektsymbol
Beispiel: (ZeigeObjektAttribute 'K1)
Typ : INSTANZ
SuperKlasse : KREIS
Attribut : K_MITTELPUNKT :(10.0 10.0)
Attribut : K_RADIUS : 40
--- ENDE OBJEKT ---- TASTE DRUECKEN -----
K1

***(ZeigeHierarchie** topclass)*
(SZH topclass)
Ausgabe der Objekthierarchie (Taxonomie) ab einer TopKlasse als strukturiertes alphanum. Listing
Argumente : *topclass* - Typ Symbol (ALOS-Klasse)
Seiteneffekte : alph. Bildschirmausgabe
Fehler : 'ZEIGEHIERARCHIE_NO_OBJEKT
Der symbolische Ausdruck *topclass* ist keine gültige ALOS-Klasse
Return : *topclass* Objektsymbol
Beispiel: (ZeigeHierarchie 'No_Polygon_Fl)
> Klasse : NO_POLYGON_FL :
Klasse : KREIS : K1
Klasse : ELLIPSE : E1
--- Ende Hierarchie ---- Taste drücken -----
No_Polygon_Fl

2.4.12. Fehlerfunktion

***(ObjErr** ErrObj ErrFun Paramliste)*
ALOS Fehlerfunktion - Ausgabe des Objektnamens bei dem ein Fehler festgestellt wurde, des Fehlernamens und einer Parameterliste. Beim Auftreten eines Fehlers kann mit ^C abgebrochen oder mit beliebiger Taste fortgesetzt werden.
ALOS verwendet nicht die Standardfehlerbehandlung von AutoLisp ***ERROR***.
Diese steht dem Anwender also weiterhin zur Verfügung.
Argumente : *Errobj* - Typ Symbol (fehlerhaftes Objekt)
ErrFun - Typ Symbol (Fehlersymbol)
Parameterlist - Typ List (Liste der Parameter mit spezifischer Bedeutung je Fehler)
Seiteneffekte : alph. Bildschirmausgabe
Fehler : keine
Return : NIL
Beispiel: (Objerr 'D5 'Alosobjekt?_NO_SYMBOL (List d5))
> Fehler in Funktion/Methode: ALOSOBJEKT?_NOSYMBOL
Fehler bei Objekt : D5
--- Parameterliste ---

```
NIL
--- Abbruch mit ^C ---
NIL
```

2.4.13 Verwaltung der Instanzliste

Die Verwaltung von Objekten in Symbolen gleichen Namens hat zwei Nachteile, zum einen ist es dadurch nicht möglich Objekte gleichen Namens zu erzeugen oder die Symbole anderweitig zu verwenden, zum anderen ist es unter AutoLISP nicht möglich Symbolnamen in Strings umzuwandeln, um diese z.B. grafisch auszugeben. Letzteren Mangel wird abgeholfen, in dem jedes Objekt über einen Standardslot '*NAME'* verfügt. Um die Objektidentität wenigstens für Instanzen herzustellen, besteht für sie die Möglichkeit, ihre Objektinformationen in generierten Symbolen ("$IAAAA" ... "$IXXXX") abzulegen. Weiterhin wird für sie eine Instanztabelle folgender Struktur geführt :

```
(  (namestr_1  (osym_11 osym_12 ... osym_1n)   ← Col 1
   (namestr_2  (osym_21 osym_22 ... osym_2n)     ....          ...
   (namestr_m  (osym_m1 osym_m2 ... osym_mn)      ← Col m
)
```

Diese Instanztabelle ist im Symbol **$ITAB0** gespeichert. Da die benutzte dynamische Tabelle mit Zugriff über Schlüsselwörter eine flexible Datenstruktur auch für andere Zwecke darstellt, wurden die entsprechenden Manipulationsfunktionen allgemein formuliert. Insbesondere besteht so die Möglichkeit, verschiedene Instanztabellen zu verwenden! Die jeweils aktuelle sollten dann auf **$ITAB0** gespeichert werden, dessen Inhalt vorher zu sichern wäre.
Alle angegebenen Funktionen arbeiten seiteneffektfrei, d.h. um die Änderungen der angegebenen Instanztabelle zu fixieren, muß der Instanztabelle der Rückgabewert der entsprechenden Funktionen mit SETQ zugewiesen werden.

***(PutToCol** Table key element)*
Ablegen eines Elementes unter einem Schlüssel *key* in der Tabelle *Table*.
Achtung : Der Schlüssel muß existieren!
Argumente : *Table* - Typ List (ASSOC-Liste)
key - Typ Atom (Schlüssel)
(im Fall der Instanztabelle Typ String)
element - Typ beliebig
Seiteneffekte : keine
Fehler : -
Return : veränderte ASSOC-Liste
Beispiel: (PutToCol $ITAB0 "BAD" '$IAACR)
> (("KEY" (...)) ... ("BAD" (BAD1 BAD2 $IAACQ **$IAACR**)) ...)

***(DelFromCol** Table key element)*
Löschen eines Elementes unter einem Schlüssel *key* in der Tabelle *Table*.
Wenn das letzte Element eines Schlüssels gelöscht wurde, wird die Spalte *key* selbst gelöscht.
Achtung : Der Schlüssel muß existieren!
Argumente : *Table* - Typ List (ASSOC-Liste)
key - Typ Atom (Schlüssel)
(im Fall der Instanztabelle Typ String)
element - Typ beliebig
Seiteneffekte : keine
Fehler : -
Return : veränderte ASSOC-Liste
Beispiel: (DelFromCol $ITAB0 "BAD" '$IAACR)
> (("KEY" (...)) ... ("BAD" (BAD1 BAD2 $IAACQ)) ...)

***(ExistCol?** Table key)*
Existiert ein Tabelleneintrag unter Schlüssel *key* in der Tabelle *Table*.
Argumente : *Table* - Typ List (ASSOC-Liste)
key - Typ Atom (Schlüssel)

(im Fall der Instanztabelle Typ String)
Seiteneffekte : keine
Return : NIL oder das entsprechende Key-Value Paar
Beispiel: (ExistCol? $ITAB0 "BAD")
> ("BAD" (BAD1 BAD2 $IAACQ))
(ExistCol? $ITAB0 "Quatsch")
> NIL

***(CreateCol** Table key)*
Erzeugen einer neuen Tabellenspalte mit dem Schlüssel *key* in der Tabelle *Table*. Als Elementeintrag existiert nur die '().
Achtung : Wenn der Schlüssel schon vorhanden ist, wird er ein zweites mal angelegt (Fehler!)
Argumente : *Table* - Typ List (ASSOC-Liste)
key - Typ Atom (Schlüssel)
(im Fall der Instanztabelle Typ String)
Seiteneffekte : keine
Fehler : -
Return : veränderte ASSOC-Liste
Beispiel: (SETQ $ITAB0 (CreateCol $ITAB0 "Duschen"))
> (("KEY" (...)) ...
("BAD" (BAD1 BAD2 $IAACQ **$IAACR**)) ...
("Duschen" ())
)

***(GetCol** Table key)*
Holen einer Tabellenspalte mit dem Schlüssel *key* aus der Tabelle *Table*.
Achtung : Der Schlüssel *key* muß existieren.
Argumente : *Table* - Typ List (ASSOC-Liste)
key - Typ Atom (Schlüssel)
(im Fall der Instanztabelle Typ String)
Seiteneffekte : keine
Fehler : -
Return : Elementliste die unter *key* gefunden wurde
Beispiel: (GetCol $ITAB0 "BAD")
> (BAD1 BAD2 $IAACQ **$IAACR**)

***(DelCol** Table key)*
Löschen einer Tabellenspalte mit dem Schlüssel *key* aus der Tabelle *Table*.
Argumente : *Table* - Typ List (ASSOC-Liste)
key - Typ Atom (Schlüssel)
(im Fall der Instanztabelle Typ String)
Seiteneffekte : -
Fehler : -
Return : veränderte ASSOC-Liste wenn *key* vorhanden war, sonst alte Liste Beispiel: (DelCol $ITAB0 "BAD")
> (("KEY" (...)) ...
("Duschen" ())
)

2.4.14 Hilfsfunktionen

Allgemeine Listenfunktionen

***(Unik** liste)*
Entfernen doppelter Elemente aus einer Liste.
Seiteneffekte : keine
Argumente : Liste
Fehler : keine
Return : Liste ohne doppelte Elemente
Beispiel: (Unik '(a b c d d a r))
> (A B C D R)

***(LDiff** liste1 liste2)*

Unsymmetrische Listendifferenz, d.h. entfernen aller Elemente von *liste1* die Mitglied von *liste2* sind.
Seiteneffekte : keine
Argumente : *liste1, liste2* Typ Liste
Fehler : keine
Return : eine Liste
Beispiel: (LDiff '(a b c d d a r) '(a c x))
> (B D D R)

***(Remove** liste ele)*
Einmaliges entfernen eines Elementes *ele* aus einer Liste.
Seiteneffekte : keine
Argumente : *liste* - Typ Liste
ele - beliebig
Fehler : keine
Return : eine Liste
Beispiel: (Remove '(a b c d d a r) 'd)
> (A B C D R)

Allgemeine ASSOC-Listenfunktionen

***(SchluesselListe** ali)*
Erzeugen der Liste aller Schlüssel einer ASSOC-Liste.
Seiteneffekte : keine
Argumente : *ali* - ASSOC-Liste
Fehler : keine
Return : Liste der Schlüssel
Beispiel: (SchluesselListe '((k1 abc) (k2 NIL)))
> (K1 K2)

***(ARemove** ASSOClist Key)*
Einmaliges entfernen eines ASSOC-Listenelementes des Schlüssels *key*
Seiteneffekte : keine
Argumente : *ASSOClist* - ASSOC-Liste
key - Atom
Fehler : keine
Return : Verkürzte ASSOC-Liste
Beispiel: (ARemove '((k1 abc) (k2 NIL)) 'k2)
> ((K1 ABC))

***(UpdProp** new key ASSOClist)*
Updaten eines ASSOC-Listenelementes des Schlüssels *key*
Seiteneffekte : keine
Argumente : *ASSOClist* - ASSOC-Liste
key - Atom
new - beliebig
Fehler : keine
Return : Veränderte ASSOC-Liste
Beispiel: (UpdProp 123 'k1 '((k1 abc) (k2 NIL)))
> ((K1 123) (K2 NIL))

***(GetProp** key ASSOClist)*
Holen der 'Daten' eines ASSOC-Listenelementes des Schlüssels *key*
Achtung : der Schlüssel *key* muß existieren!
Seiteneffekte : keine
Argumente : *ASSOClist* - ASSOC-Liste
key - Atom
Fehler : keine
Return : CADR des ASSOC-Listenelementes
Beispiel: (GetProp 'k1 '((k1 abc) (k2 NIL)))
> ABC

Allgemeine Funktionen zur Symbolmanipulation

***(GenSym**)*
Erzeugen eines Symbols mit fortlaufendem Symbolnamen ($SAAAA, $SAAAB ,..., $SZZZZ (ca. 400000 mögliche Symbole). Der entsprechende Zählerstring steht in $SYM0.
Seiteneffekte : Erzeugen eines Symbols, ändern des Zählerstrings.
Fehler : keine
Return : Symbol
Beispiel: (Gensym)
> $SAAXY

***(GenInsName**)*
Erzeugen eines Symbols mit fortlaufendem Symbolnamen ($IAAAA , $IAAAB ,..., $IZZZZ (ca. 400000 mögliche Symbole). Der entsprechende Zählerstring steht in $INST0.
Achtung: es erfolgt dabei kein automatischer Eintrag in die Instanztabelle!
Seiteneffekte : Erzeugen eines Symbols, ändern des Zählerstrings.
Fehler : keine
Return : Symbol
Beispiel: (GenInstName)
> $IAXYZ

***(GenMetName**)*
Erzeugen eines Symbols mit fortlaufendem Symbolnamen ($MAAAA, $MAAAB ,..., $MZZZZ (ca. 400000 mögliche Symbole). Der entsprechende Zählerstring steht in $MET0.
Seiteneffekte : Erzeugen eines Symbols, Ändern des Zählerstrings.
Fehler : keine
Return : Symbol
Beispiel: (GenMetName)
> $MABCD

***(Symbol?** arg)*
Predikative Funktion die prüft, ob ein *arg* mit einem Symbol belegt ist.
Seiteneffekte : keine
Argumente : beliebig
Fehler : keine
Return : T / NIL
Beispiel: (Symbol? 'a)
> T
(Symbol? 13)
> NIL

***(ExprToString** arg)*
Diese Funktion überführt einen symbolischen Ausdruck in seine 'printed-form'. Achtung: es wird eine Zwischen**datei** erzeugt (aber aus Effizienzgründen nicht gelöscht).
Seiteneffekte : Anlegen einer temporären Zwischendatei
Argumente : beliebig
Fehler : keine
Return : String der 'printed-from' von *arg*
Beispiel : (ExprToString 'a)
> "A"
(ExprToString '13)
> "13"
(ExprToString '(a (12 () ws) () k))
> "(A (12 nil WS) nil K)"

3. ALAS - AutoLisp Aggregatsystem

3.1. Zielstellung

Betrachtet man gegenwärtige CAD-Systeme, so verfügen sie (in beschränktem Maße) über die Eigenschaften,

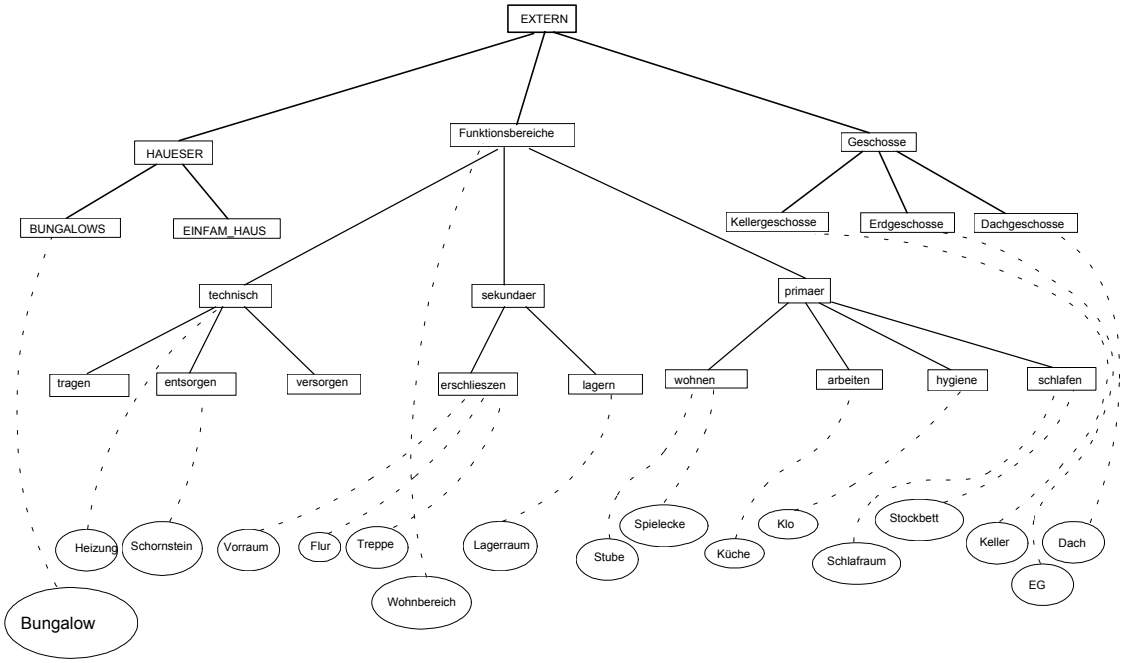
- daß sich logische oder physische Entitäten der Domäne als informationelle Einheiten handhaben lassen
- daß die Entitäten aggregiert werden können

nicht jedoch über die Fähigkeit, daß

- die Objekte sich 'Verhalten', d.h. daß sie agieren und interagieren können
- die Objekte Instanzen eines hierarchisch geordneten, begrifflichen Systems sind (Taxonomie)

Dieser Mangel entsteht nicht eigentlich durch Mängel der (objektorientierten) Implementationswerkzeuge dieser Systeme, da diese an sich ja gerade die letztgenannten Fähigkeiten besitzen, sondern aus der inkonsequenten Nutzung dieser Fähigkeiten bei der Gestaltung von CAD-Systemen.
Das Klassensystem von OO-Sprachen ist geeignet, *begriffliches Wissen* (Taxonomien) darzustellen (siehe Abb.). Nur hier ist die Vererbung als ein Kernbestandteil der OO sinnvoll erklärt! Aggregieren dagegen, als eine der wesentlichen Handlungen beim Entwerfen, ist kein Bestandteil der Grundphilosophie der OO-Programmierung. Das Zusammenstellen von Aggregationen als dem Ziel von Entwurfshandlungen und deren Repräsentation als Entitäten ist also umfassend zu unterstützen.
Bei Nutzung von OO-Werkzeugen zur Implementation entstehen die zu kreierenden Artefakte als Blätter einer Typhierarchie. Diese Darstellung ist künstlich, die Eigenschaft, daß sich Entitäten als informationelle Einheit handhaben lassen, ist nur implizit vorhanden.

Taxonomie



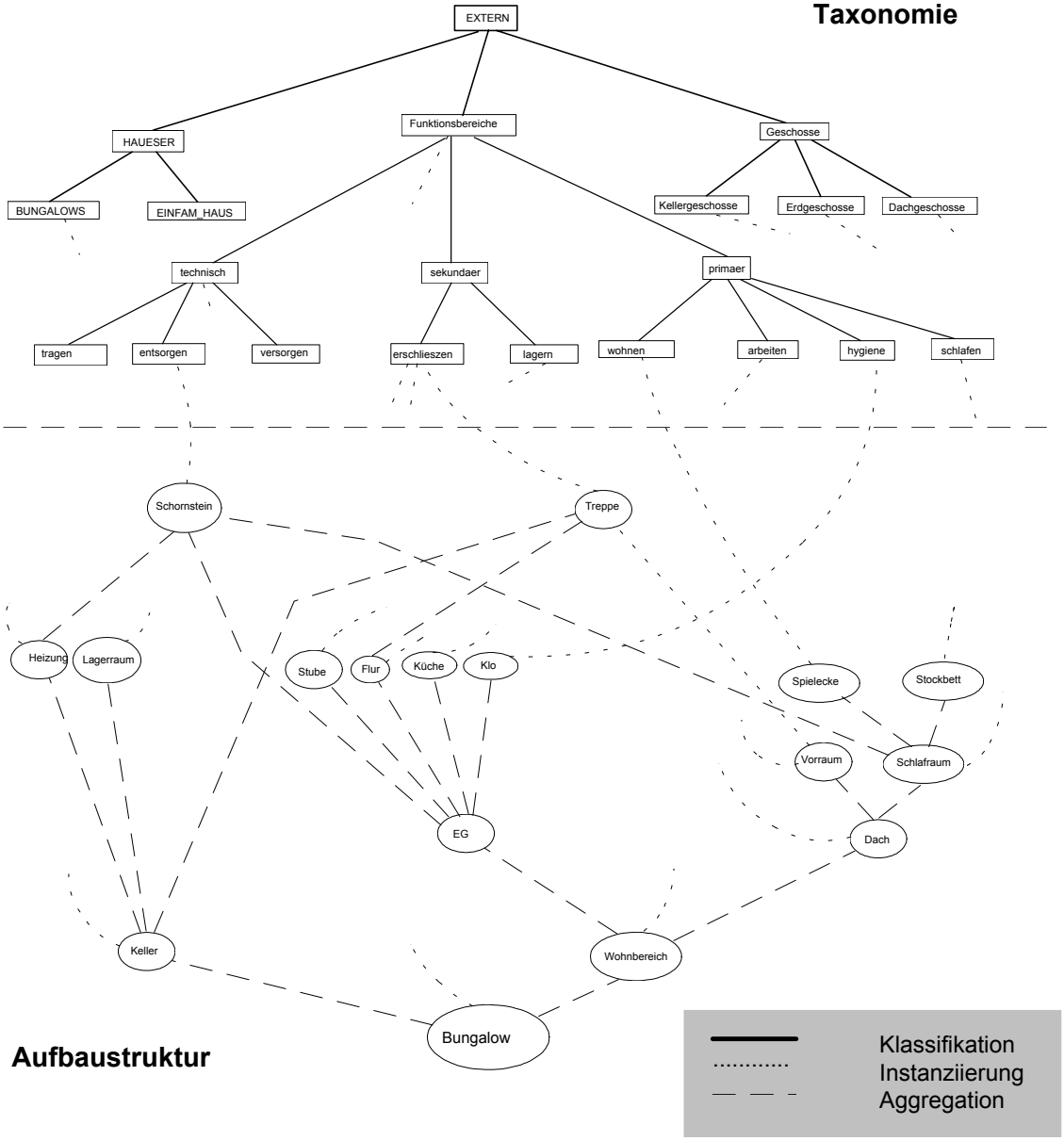
Instanzen

Wünschenswert wäre eine Darstellung, die den inneren Aufbau eines Objektes (Aufbaustruktur) aus Bestandteilen wiedergibt. Die Verwendung des Begriffes 'Subobjekt' für diese Bestandteile ist irreführend, sie werden im weiteren Teile (-objekte) genannt. Eine wünschenswerte Darstellung in o.g. Sinne zeigt die folgende Abbildung.

3.2. Begriffe

In der letzten Abbildung sind im Aggregationsgraphen entsprechend ihrer Lage zwei verschiedene Objekttypen erkennbar. Dies sind **Simplexe** und **Komplexe.**

Simplexe
Simplexe sind Instanzobjekte, die über (noch) keine Aufbaustruktur verfügen, d.h. der HAS-PART Slot enthält eine leere Liste. Sie können sehr wohl Teil von etwas sein. Sie stellen die Blätter der aktuellen Aufbaustruktur dar. Ein Simplex ist also eine Entwicklungsstufe eines Entwurfsobjektes, alle Objekte der Aggregatstruktur sind initial Simplexe.



Komplexe

Komplexe sind Instanzobjekte, die über eine Aufbaustruktur verfügen, d.h. der HAS-PART Slot enthält eine Liste mit der Angabe wenigstens eines Teilobjektes, welches Komplex oder Simplex sein kann. Komplexe können selbst wiederum Teil von etwas sein. Sie stellen die Nichtblattknoten der jeweiligen heterarchischen Aufbaustruktur dar.

Heterarchisch heißt, daß sich die Aufbaustruktur durch einem Graphen repräsentieren läßt:

- dessen Knoten Instanzobjekte sind,
- dessen gerichtete Kanten die PART-OF / HAS-PART Beziehung darstellen,
- in dem ein oder mehrere vorgängerlose Wurzelknoten existieren,
- in dem jeder Knoten erreichbar ist und
- der zyklenfrei ist.

Weiterhin gilt für **ALAS** das
- jede Kante nur einmal existiert.

AutoCAD selbst gestattet das bottom-up Aggregieren von Zeichnungselementen über den 'BLOCK' Befehl. Die Aufbaustruktur der Zeichnungselemente ist jedoch nur intern verfügbar (Explodieren funktioniert). Wurde ein solches Komplexobjekt (Block) allerdings aufgelöst, so läßt der sich nur durch wiederum explizite Angabe seiner Bestandteile erneut aggregieren. In **ALAS** sind alle Objekte der Aufbaustruktur eigenständige Objekte und als solche verfügbar. Da i.A. aber mit den Stufen der Aufbaustruktur auch ein Abstraktionsprozeß einher geht, läßt sich der Abstraktionsgrad im Entwurfsprozeß flexibel anpassen. Objekte können nach Bedarf als Ganzes oder als aus Teilen bestehend betrachtet werden.

3.3. Datenstrukturen des ALAS

IN **ALAS** ist die HAS-PART und PART-OF Relation nur für Instanzen der Klasse und Subklassen von ‘EXTERN erklärt, da nur hier die entsprechenden Slots verfügbar sind. Die Relationen sind als Liste von Verweisen auf Objekte gespeichert.

Alle Instanzen der Klasse EXTERN verfügen über die Slots

(PART_OF (...)) Verweis auf die Komplexobjekte, an deren Aufbau das Objekt beteiligt ist
(HAS_PART (...)) Verweis auf die Teilobjekte

Alle Funktionen von **ALAS** beschäftigen sich nur mit den Kanten des Graphen der Aufbaustruktur. Es werden keine Instanzen erzeugt oder gelöscht! Die entsprechenden Objekte müssen bereits existieren.
Im weiteren folgt die Beschreibung aller verfügbaren Funktionen des Aggregatsystems. Zur genaueren Analyse sei wiederum auf den Quelltext verwiesen. Alle zur Benutzung des Aggregatsystems benötigten Slots werden bereits in *'AOINIT.LSP'* angelegt.

3.4. Modulbeschreibung

Modul : **AABASIS.LSP**
Beschreibung : Der Modul definiert die Basisfunktionen zum Ein- und Ausbau von Teilen, d.h. zum Erweitern- und Verkürzen der Aufbaustruktur. Es werden keine Objekte gelöscht oder erzeugt!

```
Exportierte Funktionen :
      (DEFUN BauEin (teil ortlist)  ...  )
      (DEFUN SBE (teil ortlist)  ...  )
      (DEFUN BauAus (teil ortlist)  ...  )
      (DEFUN SBA (teil ortlist)  ...  )
```

Modul : **AAEDIT.LSP**
Beschreibung : Der Modul definiert die Funktionen zum Ein- und Ausbau von Teilen, d.h. zum Erweitern- und Verkürzen der Aufbaustruktur. Es werden keine Objekte gelöscht oder erzeugt!

```
Exportierte Funktionen :
      (DEFUN VerschiebeTeil
              (teil altklxlist ausbfkt newklxlist einbfkt)    ... )
      (DEFUN SVT (teil altklxlist newklxlist) ... )
      (DEFUN Explodiere (komplex ausbfkt inkomplex einbfkt) ... )
      (DEFUN SEXP (teil) ... )
      (DEFUN Ersetze (teillist ausbfkt einbfkt1 ausbklxlist einbklx
              einbfkt2) ... )
      (DEFUN SERS (teillist ausbfkt einbfkt1 ausbklxlist einbklx
              einbfkt2) ... )
      (DEFUN Entferne (teil ausbfkt) ... )
      (DEFUN SENT (teil) ... )
      (DEFUN EntferneStrikt (teil ausbfkt) ... )
      (DEFUN SENTS (teil) ... )
```

Modul : **AAANALY.LSP**
Beschreibung : Der Modul definiert die Funktionen zur Analyse der Aufbaustruktur.

```
Exportierte Funktionen :
      (DEFUN Teile (komplex)  ...  )
      (DEFUN ST (komplex)  ...  )
      (DEFUN AlleTeile (komplex)  ...  )
      (DEFUN SAT (komplex)  ...  )
      (DEFUN ExklusiveTeile (komplex)  ...  )
      (DEFUN SEXT (komplex)  ...  )
      (DEFUN AlleExklusiveTeile (komplex)  ...  )
      (DEFUN SAEXT (komplex)  ...  )
      (DEFUN STV (teil)  ...  )
      (DEFUN Simplexe (komplex)  ...  )
      (DEFUN SSIM (komplex)  ...  )
      (DEFUN AlleSimplexe (komplex)  ...  )
      (DEFUN SASIM (komplex)  ...  )
```

```
(DEFUN AlleExklusiveSimplexe (komplex)  ...  )
(DEFUN SAEXSI (komplex)  ...  )
(DEFUN TeilVon (teil)  ...  )
(DEFUN AlleWurzeln (objlist) ... )
```

Modul : **AAPRED.LSP**
Beschreibung : Der Modul definiert die predikativen Funktionen zur Analyse der Aufbaustruktur.

```
Exportierte Funktionen :
(DEFUN Komplex? (obj)  ...  )
(DEFUN SKLX? (obj)  ...  )
(DEFUN Simplex? (obj)  ...  )
(DEFUN SSIM? (obj)  ...  )
(DEFUN TeilVon? (teil)  ...  )
(DEFUN STV? (teil)  ...  )
(DEFUN Exklusiv? (teil)  ...  )
(DEFUN SEX? (obj)  ...  )
```

Modul : **AALOOP.LSP**
Beschreibung : Der Modul definiert Funktionen zur zyklischen Anwendung von Funktionsausdrücken auf Teile des Aufbaustrukturgraphen

```
Exportierte Funktionen :
(DEFUN Kumuliere  (partsym komplex expression)  ... )
(DEFUN SK  (partsym komplex expression)  ... )
```

Modul : **AAZEIG.LSP**
Beschreibung : Der Modul definiert eine Funktion zur alphanum. Anzeige der AufbauSTRUKTUR.

```
Exportierte Funktionen :
(DEFUN ZeigeKomplex (obj)  ...  )
(DEFUN SZKLX (obj)  ...  )
```

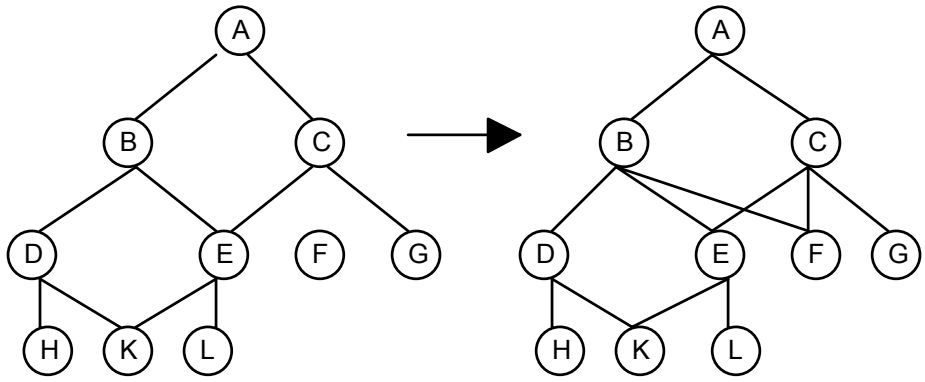
Modul : **AAORG.LSP**
Beschreibung : Der Modul definiert die Funktionen zum Löschen aller Kanten des Strukturgraphen sowie der Überprüfung der korrekten Verweiseinträge in HAS-PART und PART-OF (Symmetrie)

```
Exportierte Funktionen :
(DEFUN ResetALAS ()  ...  )
(DEFUN FehlerALAS? (top)  ...  )
```

3.5. Referenz

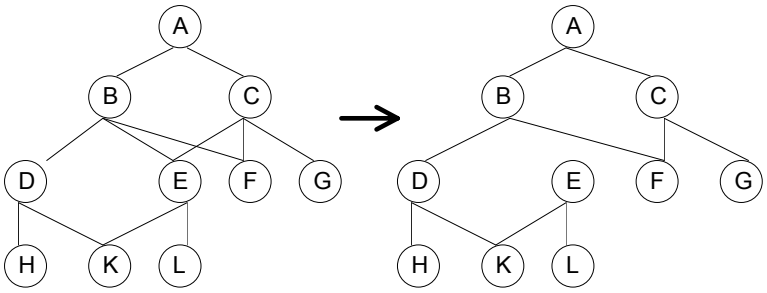
3.5.1. Basisfunktionen

***(BauEin** teil ortlist*)
(SBE teil ortlist)
Eine existierende Instanz *teil* wird Teil aller Instanzen in *ortlist*, d.h. sie wird in den Aufstrukturgraphen integriert. Es werden keine Objekte generiert, sondern lediglich eine Kante des Graphen! *teil* kann mehrmals eingebaut werden (Heterarchie). Es wird bei *teil* der Slot 'PART-OF' ergänzt und bei allen Instanzen in *ortlist* der Slot 'HAS-PART'. Der Monitor 'BeiEinbau' feuert wenn vorhanden. In der Kurzform *SBE* wird auf Fehlertest verzichtet. Der Monitor 'BeiEinbau' feuert dort **nicht**. Bei Bedarf kann jedoch wie folgt explizit getriggert werden :
(IF (Slot? 'BEIEINBAU *teil*) (SendeNachricht *teil* 'BEIEINBAU (LIST *ort*)))
Argumente : *teil* - Typ Symbol (ALOS-Instanz)
ortlist - Liste von Symbol (ALOS-Instanz)
Seiteneffekte : die ASSOC-Listen von aller Instanzen von *ortlist* und *teil* werden geändert
Fehler : Der symbolischen Ausdrücke *teil* oder einer Instanz aus *ortlist* sind keine gültigen ALOS-Instanzen
teil ist bereits Teil von einer Instanz aus *ortlist*
Return : neue Komplexliste (PART-OF) von *teil*
Beispiel: (BauEin 'Treppe '(Keller $IAABC))
> (KELLER ETTAGE1 $IAABC)



BauEin('F '(B C))

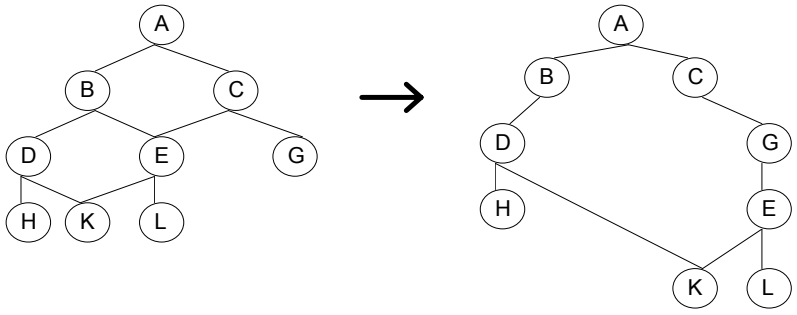
***(BauAus** teil ortlist*)
(SBA teil ortlist)
Eine existierende Instanz *teil* wird als Teil aus allen Instanzen in *ortlist* entfernt, d.h. sie wird aus dem Aufbaustrukturgraphen entfernt. Es werden keine Objekte gelöscht, sondern lediglich eine Kante des Graphen! Es wird bei *teil* der Slot 'PART-OF' verkürzt und bei allen Instanzen aus *ortlist* der Slot 'HAS-PART'.
Der Monitor 'BeiAusbau' feuert wenn vorhanden. In der Kurzform *SBE* wird auf Fehlertest verzichtet. Der Monitor 'BeiAusbau' feuert dort **nicht**. Bei Bedarf kann jedoch wie folgt explizit getriggert werden:
(IF (Slot? 'BEIAUSBAU *teil*) (SendeNachricht *teil* 'BEIAUSBAU (LIST *ort*)))
Argumente : *teil* - Typ Symbol (ALOS-Instanz)
ortlist - Liste von Symbol (ALOS-Instanz)
Seiteneffekte : die ASSOC-Listen von allen Instanzen in *ortlist* und in *teil* werden geändert
Fehler: Der symbolischen Ausdrücke *teil* oder eine Symbol aus *ortlist* sind keine gültigen ALOS-Instanzen, *teil* ist nicht Teil aller Instanzen in *ortlist*
Return : neue Komplexliste (PART-OF) von *teil*
Beispiel: (BauAus 'Treppe '(Keller 'EG)
> ($IAABC)



BauAus('E '(B C))

3.5.2. Funktionen zum Editieren der Aufbaustruktur

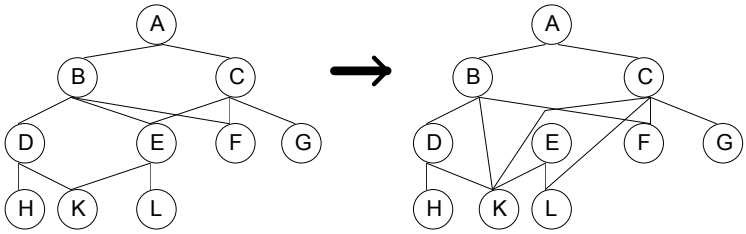
***(VerschiebeTeil** teil altklxlist ausbfkt neuklxlist einbfkt*)
(SVT teil altklxlist neuklxlist)
Eine existierende Instanz *teil* wird aus den Komplexobjekten in *altklxlist* entfernt und wird Teil der Instanzen in *neuklxlist*. Es werden keine Objekte generiert oder gelöscht, sondern lediglich Kanten des Graphen!
Es wird bei *teil* der Slot 'PART-OF' verändert und bei allen Objekten in *neuklxlist* und *altklxlist* der Slot 'HAS-PART'.
Die Parameter *einbfkt* und *ausbfkt* geben an, welche Funktionen zum Ein- bzw. Ausbau verwendet werden. Damit läßt sich das Feuern der Monitore 'BeiEinbau' und 'BeiAusBau' flexibel gestalten, sofern sie überhaupt belegt sind. Wird *SBE* (oder NIL) oder SBA (oder NIL) angegeben, so feuern die entsprechenden Monitore nie. Insbesondere lassen sich hier auch selbstdefinierte Ein- bzw. Ausbaufunktionen angeben!
Argumente : *teil* - Typ Symbol (ALOS-Instanz)
neuklxlist - Typ Liste von Symbol (ALOS-Instanzen)
neuklxlist - Typ Liste von Symbol (ALOS-Instanzen)
einbfkt - Typ Symbol (Name der Einbaufunktion oder NIL)
ausbfkt - Typ Symbol (Name der Ausbaufunktion oder NIL)
Seiteneffekte : Die ASSOC-Listen von *teil* und allen Instanzen in *altklxlist* und *neuklxlist* werden geändert.
Fehler : - Der symbolische Ausdruck in *teil* oder eines Elementes von *neuklxlist* oder *altklxlist* ist keine gültige ALOS-Instanz
- *teil* ist bereits Teil einer Instanz von *neuklxlist*
- *teil* ist nicht Teil aller Instanz von *altklxlist*
- *ausbfkt* oder *einbfkt* enthält keine Liste von der Struktur einer LAMBDA-Liste.
Return : neuer 'PART-OF'-Slot von *teil*
Beispiel: (VerschiebeTeil 'WAND11 '(RAUM1 RAUM2) NIL '(RAUM5) 'BauEin)
> (RAUM4 RAUM5)



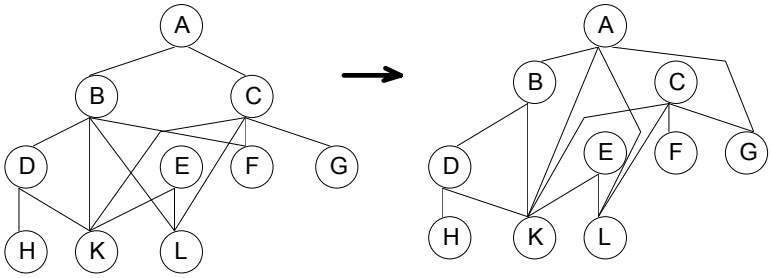
(Verschiebe 'E '(B C) NIL '(G) 'SBE)

***(Explodiere** komplex ausbfkt inkomplexlist einbfkt)*
(SEXP komplex ausbfkt inkomplex einbfkt)
In allen Komplexen in *inkomplexlist* wird der Knoten *komplex* durch seine Teile ersetzt. Er gehört dann selbst zu keinem Komplex in *inkomplex* mehr. Die Aufbaustruktur von *komplex* bleibt unverändert. Geändert werden die 'HAS-PART' Slots aller Komplexe von *inkomplexlist* sowie die 'PART-OF'-Slots von *komplex* und aller Teile von *komplex.*

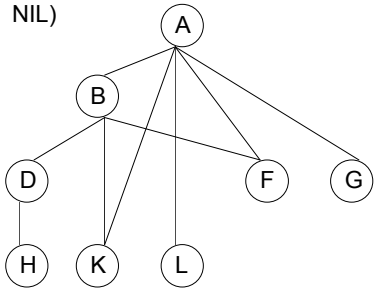
Es werden keine Objekte gelöscht, sondern lediglich Kanten des Graphen!
Die Monitore 'BeiEinBau' und 'BeiAusBau' feuern in Abhängigkeit von der verwendeten Einbaufunktion *einbfkt* und Ausbaufunktion *ausbfkt*, sofern die Monitore belegt sind.
Argumente : *komplex* - Typ Symbol (ALOS-Instanz)
inkomplexlist - Typ Liste von Symbol (ALOS-Instanzen)
einbfkt - Typ Symbol (Name der Einbaufunktion oder NIL)
ausbfkt - Typ Symbol (Name der Ausbaufunktion oder NIL)
Seiteneffekte : Die ASSOC-Listen von *komplex* , allen Instanzen in *inkomplexlistlist* und aller Teile *komplex* werden geändert.
Fehler : Der symbolischen Ausdrücke in *inkomplexlistlist* oder *komplex* sind keine gültige ALOS-Instanzen.
....
Return : T
Beispiel: (Explodiere 'EG)
> (KELLER DACH RAUM1 RAUM2 RAUM3)



Explodiere('E NIL '(B C) 'SBE)



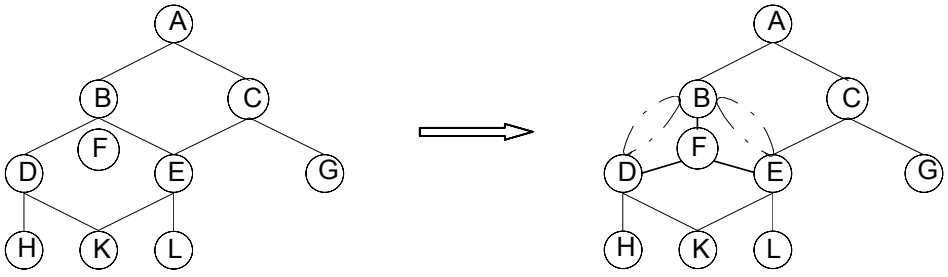
Explodiere('C 'BauAus '(A) NIL)



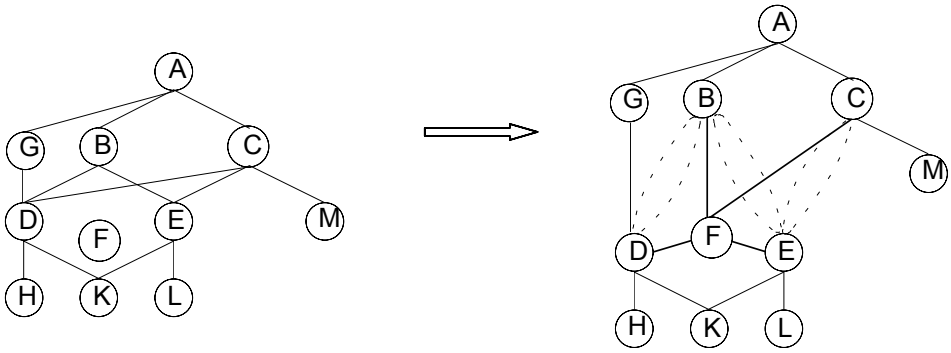
von A sichtbarer Strukturgraph

***(Ersetze** teilelist ausbfkt einbfkt1 ausbklxlist einbklx einbfkt2)*
(SERS teilelist ausbklxlist einbklx)
Die Knoten der Aufbaustruktur, die die Instanzen *teilelist* darstellen, werden in einem neuen Knoten *einbklx* aggregiert. Dazu werden sie mittels *ausbfkt* in allen Komplexen aus *ausbklxlist* ausgebaut und mit *einbfkt1* in *einbklx* eingebaut.
einbklx selbst wird mit *einbfkt2* in alle Komplexe aus *ausbklxlis* als Teil eingebaut. Alle Instanzen müssen erzeugt sein !
Der Monitor 'BeiAusBau und 'BeiEinBau feuern (falls sie überhaupt belegt sind) in Abhängigkeit von der verwendeten Ausbaufunktionen *ausbfkt*, *einbfkt1* und *einbfkt2*.
Argumente : *teilelist* - Typ Liste von Symbol (ALOS-Instanzen)
ausbfkt - Typ Symbol (dessen Inhalt muß eine ausführbare Ausbaufunktion sein).
einbfkt1 und *einbfkt2* - Typ Symbol (deren Inhalt müssen ausführbare Einbaufunktionen sein).
Seiteneffekte : die ASSOC-Listen der betroffenen Instanzen werden geändert.
Fehler : Ein symbolischer Ausdruck aus *teilelist* ist keine gültige ALOS-Instanz
...
Ein symbolischer Ausdruck aus *ausbklxlist* ist keine gültige ALOS-Instanz
...
Der symbolische Ausdruck *einbklx* ist keine gültige ALOS-Instanz
...

Return : T / NIL
Beispiel: (Ersetze '(D E) NIL 'SBE '(B) 'F 'BeiEinBau)
> T
(Ersetze '(D E) NIL 'SBE '(B C) 'F 'NIL)
> T

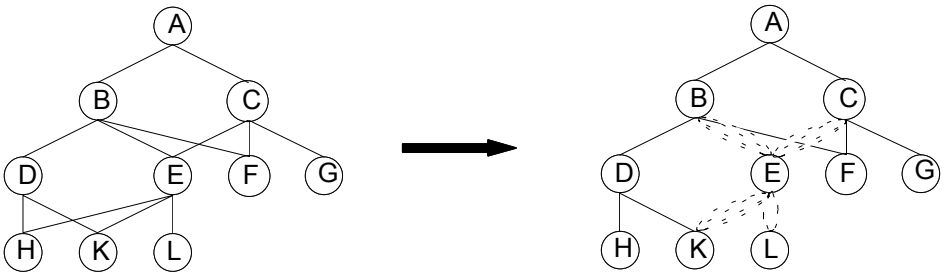


(Ersetze '(D E) NIL 'SBE '(B) 'F 'SBE)



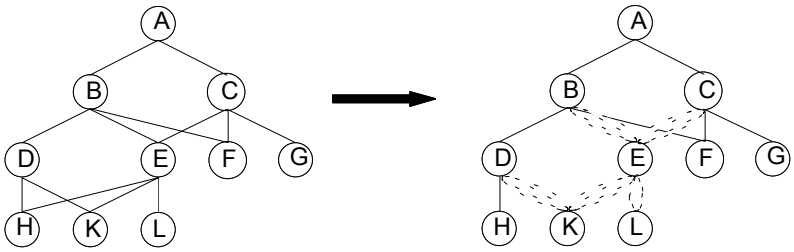
(Ersetze '(D E) NIL 'SBE '(B C) 'F 'BauEin)

***(Entferne** teil ausbfkt)*
(SENT teil)
Der Knoten der Aufbaustruktur, den die Instanz *teil* darstellt, wird entfernt sowie alle dessen direkten und indirekten Teile, sofern sie nicht Teil eines nicht zu entfernenden Objektes sind. (siehe Abb.)
Es werden keine Objekte gelöscht, sondern lediglich eine Kante des Graphen!
Aller Verweise in 'PART-OF' und 'HAS-PART'-Slots auf Objekte, die zu entfernen sind, werden gelöscht.
Der Monitor 'BeiAusBau' feuert (falls er überhaupt belegt ist) in Abhängigkeit von der verwendeten Ausbaufunktion *ausbfkt*.
Argumente : *teil* - Typ Symbol (ALOS-Instanz)
ausbfkt - Typ Symbol (dessen Inhalt muß eine ausführbare Ausbaufunktion sein).
Seiteneffekte : Die ASSOC-Listen von *teil* sowie betroffener Teilobjekte werden geändert.
Fehler : Der symbolische Ausdruck *teil* oder eines seiner Teile ist keine gültige ALOS-Instanz
...
Return : Liste der Teilobjekte von *teil*, die nicht gelöscht wurden, weil sie noch Teil anderer Komplexobjekte waren.
Beispiel: (Entferne 'EG)
> (TREPPE SCHORNSTEIN)



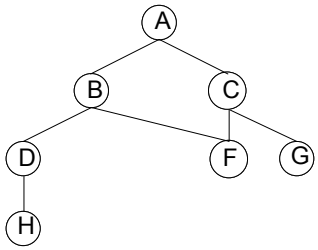
(Entferne 'E 'SBA '(B C))

***(EntferneStrikt** teil ausbfkt)*
(SENTS teil)
Der Knoten der Aufbaustruktur den die Instanz *teil* darstellt wird entfernt sowie alle dessen direkten und indirekten Teile, auch die, die Teil eines nicht zu entfernenden Objektes sind! (siehe Abb.)
Es werden keine Objekte gelöscht, sondern lediglich Kanten des Graphen! Alle Verweise in 'PART-OF' und 'HAS-PART'-Slots auf Objekte, die zu entfernen sind, werden gelöscht.
Der Monitor 'BeiAusBau' feuert (falls er überhaupt belegt ist) in Abhängigkeit von der verwendeten Ausbaufunktion *ausbfkt*.
Argumente : *teil* - Typ Symbol (ALOS-Instanz)
ausbfkt - Typ Symbol (dessen Inhalt muß eine ausführbare Ausbaufunktion sein.
Seiteneffekte : Die ASSOC-Listen von *teil* sowie betroffener Teilobjekte und des Komplexes von *teil* werden geändert.
Fehler : Der symbolische Ausdruck *teil* oder eines seiner Teile ist keine gültige ALOS-Instanz.
...
Return : Liste der Teilobjekte von *teil*, die gelöscht wurden, obwohl sie noch Teil anderer Komplexobjekte waren.
Beispiel: (Entferne 'EG)
> (TREPPE SCHORNSTEIN)



(EntferneStrikt 'E 'SBA '(B C))

von A sichtbarer Strukturgraph



3.5.3. Funktionen zum Analysieren der Aufbaustruktur

Alle nachfolgend beschriebenen Funktionen arbeiten seiteneffektfrei!

***(Teile** komplex*)
(ST komplex)
Liefert die Liste aller direkten Teile von *komplex*.
Argumente : *komplex* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *komplex* ist keine gültige ALOS-Instanz.
...
Return : Liste der Teilobjekte von *komplex* ('HAS-PART'-Slot) Beispiel: (Teile 'EG)
> (SCHORNSTEIN STUBE FLUR KÜCHE KLO)

***(ExklusiveTeile** komplex*)
(SEXT komplex)
Liefert die Liste aller direkten Teile von *komplex*, die zu keinem anderen Komplex gehören.
Argumente : *komplex* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *komplex* ist keine gültige ALOS-Instanz.
...
Return : Liste der Teilobjekte von *komplex* (HAS-PART Slot), soweit sie nicht zu anderen Komplexen gehören.
Beispiel: (ExklusiveTeile 'EG)
> (STUBE FLUR KÜCHE KLO)

***(AlleTeile** komplex*)
(SAT komplex)
Rückgabe der linearen Liste aller Teilobjekte von *obj*
Argumente : *komplex* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *komplex* ist keine gültige ALOS-Instanz
...
Return : Liste aller Teilobjekte von *komplex*
Beispiel: (AlleTeile 'EG)
> (SCHORNSTEIN STUBE FLUR TREPPE KÜCHE KLO)

***(AlleExklusiveTeile** komplex*)
(SAEXT komplex)
Rückgabe der linearen Liste aller Teilobjekte von *obj* die zu keinem anderem Komplex gehören.
Argumente : *komplex* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *komplex* ist keine gültige ALOS-Instanz.
...
Return : Liste aller Teilobjekte von *komplex*, soweit sie keinem anderem Komplex angehören.
Beispiel: (AlleExklusivenTeile 'EG)
> (STUBE FLUR KÜCHE KLO)

***(Simplexe** komplex*)
(SSIM komplex)
Rückgabe der Liste der direkten Simplexe von *obj*
Argumente : *komplex* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *komplex* ist keine gültige ALOS-Instanz.
...
Return : Liste der direkten Teilobjekte von *komplex*, die Simplexe sind
Beispiel: (Simplexe 'EG)
> (SCHORNSTEIN STUBE KÜCHE KLO)

***(ExklusiveSimplexe** komplex*)
(SEXSIM komplex)
Rückgabe der Liste der direkten Simplexe von *obj*, die exklusiv zu *komplex* gehören.
Argumente : *komplex* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *komplex* ist keine gültige ALOS-Instanz.
...
Return : Liste der direkten Teilobjekte von *komplex*, die Simplexe sind und nur zu *komplex* gehören
Beispiel: (ExklusiveSimplexe 'EG)
> (STUBE KÜCHE KLO)

***(AlleSimplexe** obj*)
(SASIM komplex)
Rückgabe der linearen Liste aller Simplexe von *obj*
Argumente : *komplex* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *komplex* ist keine gültige ALOS-Instanz
...
Return : lineare Liste aller Teilobjekte von *komplex*, die Simplexe sind
Beispiel: (AlleSimplexe 'EG)
> (SCHORNSTEIN STUBE TREPPE KÜCHE KLO)

***(AlleExklusiveSimplexe** komplex*)
(SAEXSI komplex)
Rückgabe der linearen Liste aller Simplexe von *komplex*, die nicht Teil eines anderen Komplexes sind.
Argumente : *komplex* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *komplex* ist keine gültige ALOS-Instanz.
...
Return : lineare Liste aller Teilobjekte von *komplex*, die Simplexe sind
Beispiel: (AlleExklusivenSimplexe 'EG)
> (STUBE KÜCHE KLO)

***(TeilVon** inst*)
(STV inst)
Rückgabe der Liste der Komplexe, deren Teil *inst* ist.
Argumente : *inst* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *inst* ist keine gültige ALOS-Instanz.
...
Return : Liste Komplexen, deren Teil *inst* ist.
Beispiel: (TeilVon 'EG)
> (WOHNBEREICH)

***(TeilVonListe** inst*)
(STVLI inst)
Rückgabe der linearen Liste der Komplexe, deren Teil *inst* direkt oder indirekt ist.
Argumente : *inst* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *inst* ist keine gültige ALOS-Instanz
...
Return : lineare Liste aller Komplexe, deren Teil *inst* direkt oder indirekt ist.
Beispiel: (TeilVonListe 'EG)
> (WOHNBEREICH BUNGALOW)

***(AlleWurzeln** inst)*
(SAWU inst)
Rückgabe der Liste der Komplexe, deren Teil *inst* direkt oder indirekt ist und die selbst nicht Teil eines Komplexes sind.
Argumente : *inst* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *inst* ist keine gültige ALOS-Instanz
...
Return : Liste von Komplexen
Beispiel: (AlleWurzeln 'EG)
> (BUNGALOW)

3.5.4. Prädikative Funktionen

***(Simplex?** inst)*
(SSLX? inst)
Ist *inst* ein Simplex der Aufbaustruktur ? (T/NIL)
Argumente : *inst* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *inst* ist keine gültige ALOS-Instanz
...
Return : T/NIL
Beispiel: (Simplex? 'EG)
> nil
(Simplex? 'KLO)
> T

(Komplex?** inst**)
(SKLX? inst)
Ist *inst* ein Komplex der Aufbaustruktur? (T/NIL)
Argumente : *inst* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *inst* ist keine gültige ALOS-Instanz
Return : T/NIL
Beispiel: (Komplex? 'EG)
> T
(Komplex? 'WC)
> NIL

Exklusiv? inst**)**
(SEX? inst)
Ist *inst* exklusives Teil eines Komplexes oder gehört es zu mehreren Komplexen ? (T/NIL)
Argumente : *inst* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *inst* ist keine gültige ALOS-Instanz
...
Return : T/NIL
Beispiel: (Exklusiv? 'WC)
> T
(SEX? 'Treppe)
> NIL

(ExklusivVon? inst komplex**)**
(SEXV? inst)
Ist *inst* ein direktes oder indirektes exklusives Teil von *komplex*.
Argumente : *inst* - Typ Symbol (eine ALOS-Instanz)
komplex - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *inst* ist keine gültige ALOS-Instanz
Der symbolische Ausdruck *komplex* ist keine gültige ALOS-Instanz
Return : T/NIL
Beispiel: (ExklusivVon? 'WC 'Haus)
> NIL
(SEXV? 'Treppe 'EG)
> T

3.5.5. Anzeigefunktionen (alphanumerisch)

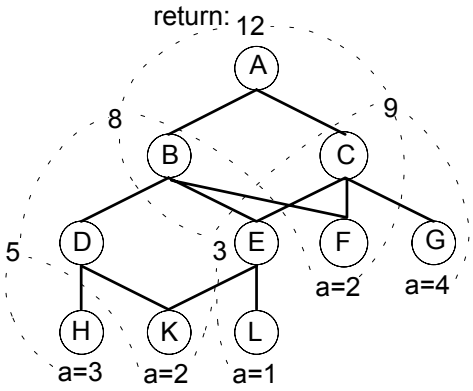
(ZeigeKomplex *inst* **)**
(SZKLX inst)
Strukturiertes Listen (alphanumerisch) der Aufbaustruktur von *inst*
Da es sich um eine Heterarchie handelt, werden Teilbäume der Aufbaustruktur die bereits gedruckt wurden, nicht noch einmal gedruckt und mit einem * versehen.
Argumente : *inst* - Typ Symbol (eine ALOS-Instanz)
Fehler : Der symbolische Ausdruck *inst* ist keine gültige ALOS-Instanz
...
Return : T
Beispiel: (ZeigeKomplex 'WOHNBEREICH)

```
WOHNBEREICH
   EG
      SCHORNSTEIN
      STUBE
      FLUR
         TREPPE
      KÜCHE
      KLO
   DACH
      VORRAUM
         TREPPE *
      SCHLAFRAUM
         SCHORNSTEIN *
         SPIELECKE
         STOCKBETT
```

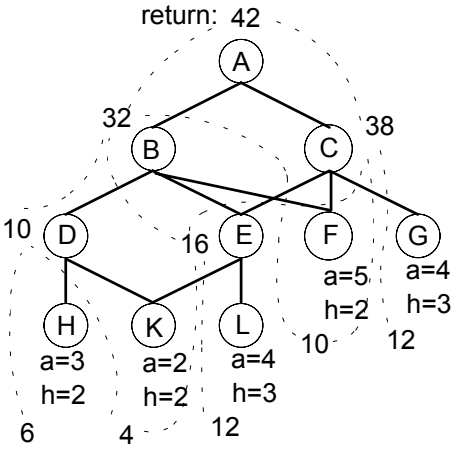
3.5.6. Spezielle Schleifenfunktionen über Aufbaustrukturen

***(Kumuliere** teilsym komplex ausdruck)*
(SKUM teilsym komplex ausdruck)
Die Funktion dient zur rekursiven Berechnung eines Wertes über alle Komplexe, die direkt oder indirekt Teil von *komplex* sind. Hierzu wird *teilsym* an alle Teile eines Komplexes gebunden und dann *ausdruck* ausgewertet. Dieses Verfahren wird rekursiv für alle Teile von *komplex* wiederholt und die Ergebnisse dieser Auswertung werden aufsummiert (kumuliert).
Teile, die über verschiedene Wege von *komplex* aus erreicht werden können, werden nicht mehrfach berechnet!
Alle in *ausdruck* genutzten Slots müssen in allen Simplexen, an die *teilsym* gebunden wird, verfügbar sein!
Argumente: *teilsym* - Typ Symbol (lokales Symbol das in *ausdruck* an alle Teile des bearbeiteten Komplexes gebunden wird)
komplex - Typ Symbol (ALOS-Instanz)
Ist der Topknoten der Aufbaustruktur, bis zu dem kumuliert wird.
ausdruck- gequoteter Funktionenausdruck
Fehler : Der symbolische Ausdruck *komplex* ist keine gültige ALOS-Instanz
...
Return : kumulierter Gesamtwert
Beispiel:

```
(Kumuliere 'PART  'BUNGALOW  '( (* (Hole PART 'Hoehe)
                                   (SendeNachricht PART 'Flaeche '())
)                                   ) )
355
(Kumuliere 'PART  'A '( (* (Hole PART 'a) (Hole PART 'h) ) )     )
42
(Kumuliere 'PART 'A '( (Hole PART 'a) ))
12
```




```
(Kummuliere 'PART 'A
   '( (Hole PART 'a) )
)
```

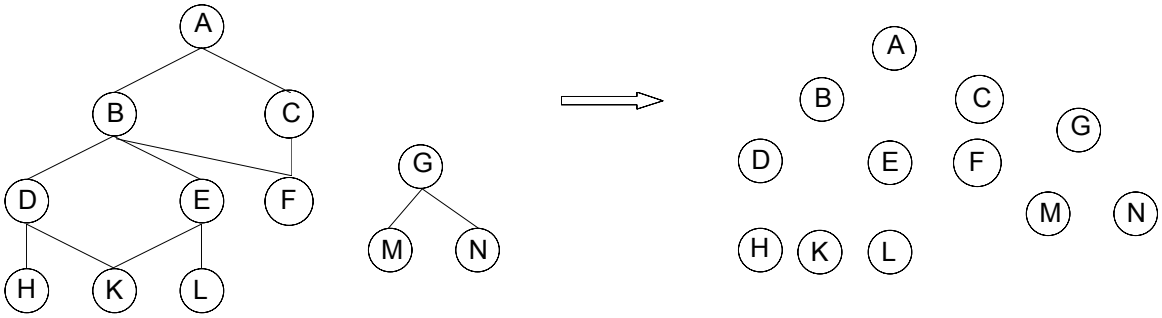



```
(Kummuliere 'PART 'A
   '( (* (Hole PART 'h)
      (Hole PART 'a)
)   ) )
```

3.5.7. Organisationsfunktionen

Die Funktionen sind Funktionen, die hauptsächlich in der Entwicklungsphase dem Korrektheitstest und dem Rücksetzen des Gesamtsystems dienen. Ihr Funktionsumfang kann in neuen Versionen von **ALAS** erweitert werden.

***(ResetALAS*)**
Alle Kanten des Aufbaustrukturgraphen bei allen Instanzen von 'EXTERN oder deren Subklassen werden gelöscht.
Argumente : -
Seiteneffekte : alle 'PART-OF' und 'HAS-PART'-Slots aller direkten und indirekten Instanzen von 'EXTERN werden auf NIL gesetzt
Fehler : -
Return : T
Beispiel: (ResetALAS)
> T



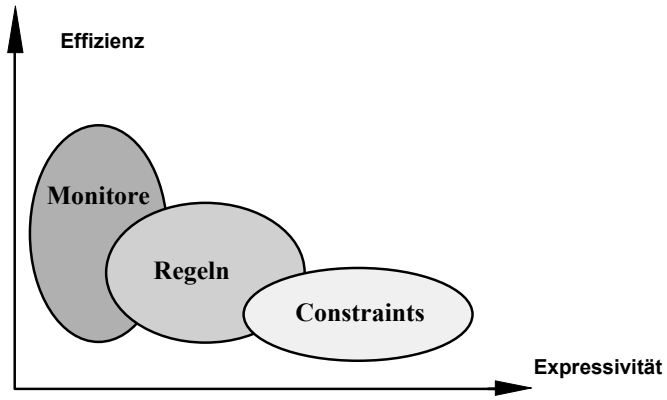
(ResetALAS)

***(FehlerhaftALAS?** top)*
Die Funktion prüft, ob alle Kanten des Strukturgraphen ab *top* symmetrisch vorhanden sind und ob ein Objekt mehrfach direktes Teilobjekt eines Komplexes ist.
Argumente : *top* - Typ Symbol (ALOS-Instanz)
Seiteneffekte : -
Fehler : *top* oder eines seiner Teilobjekte ist keine ALOS-Instanz.
Return : '() - wenn alle Kanten symmetrisch vorhanden sind
'((part komplex) ...) - Liste fehlerhafter Aufbaubeziehungen
(Bem.: Alles außer nil ist t)
Beispiel: (FehlerhaftALAS? 'A)
> nil

4. Weiterführende Module

4.1. Propagation in ALRES, ALMS

Im Allgemeinen existieren für Objekte in komplexen Modellen vielfältige Beziehungen verschiedener Typen. So haben Entwurfshandlungen oft sehr komplexe, weitreichende und tief geschachtelte Wirkungen (siehe vorn), deren Ausbreitung/ Fortpflanzung (lat. Propagation) vom Nutzer nur selten überschhaut werden. Zur Beschreibung solcher Auswirkungen stehen verschiedene Mittel zur Verfügung die sich in Expressivität, Einsatzzweck, Art der Propagation und damit Effizienz unterscheiden.



Ziel ist es, dem Nutzer Repräsentationen an die Hand zu geben, mittels denen er die Wirkungen von Entwurfshandlungen adäquat beschreiben kann. Nach Möglichkeit sollte die Beschreibung so formalisiert sein, daß spezifizierte Wirkungen automatisch ausgelöst (propagiert) werden können.
Dafür bieten sich verschiedene Mechanismen an, die gegenwärtig nicht Bestandteil der OOP sind. Sie sind in (manchen) KI-Shells/KI-Toolkits zu finden.
Interessanter weise bieten einige objektorientierte Modelliermethodiken (CASE-Techniken) diese Mechanismen als Beschreibungsmittel. Diese Mittel sind **Monitore, Regeln, Constraints**.

4.2. Monitore

Sie sind der effizienteste Weg zur Triggerung von Folgeaktionen, da hierzu nur ein lesender- oder schreibender Zugriff zu dem mit einem Monitor versehenen Slot nötig ist. Der Mechanismus selbst beachtet keine inhaltlichen Randbedingungen. Es sind somit sehr programmiersprachnahe Mechanismen mit geringer Expressivität. Die Aktionen selbst sind in Methoden codiert die der Klassen zu geordnet werden. Der Monitor ist als Verweis auf diese Methode realisiert, die bei entsprechender Zugriffsart zu triggern ist.

Folgende Monitore werden oft verwendet:

WHEN-ACCESS Methode wird bei lesend oder schreibend Zugriff ausgeführt

IF-NEEDED Die Methode wird bei jedem lesenden Zugriff ausgeführt, wenn der entsprechede Slot noch keine Wertebelegung hat. Im System muß also ein spezielles Symbol existieren, das einen unbekannten Wert darstellt. (in **ALOS** 'NICHTS)

BEFORE-CHANGE
AFTER-CHANGE Die Methode wird ausgeführt bevor, bzw. nachdem die Wertebelegung des überwachten Slots durchgeführt wird. Die Methode erhält als Argument den einzutragen, bzw. den alten Slotwert.

IF-INHERITED Der Monitor feuert, wenn bei einem lesenden Zugriff der (geerbte) Defaultwert zurück gegeben wird

AFTER-ACCESS
BEFORE-ACCESS
AFTER-READ
BEFORE-READ (anlog oben).

Durch die Homogenität des Slotbegriffes sind mit den letzgenannten Monitoren Pre- und Postprozesse abbildbar, wie sie z.B. aus OMT (siehe J.RUMBAUGH) bekannt sind.
Eine interessante Erweiterung stellen Monitore dar, die nicht einem Slot zugeordnet sind, sondern einer Klasse. Sie werden getriggert, wenn Instanzen erzeugt oder gelöscht werden, bzw. in eine Aufbaustruktur ein- oder ausgebaut werden.

Diese Monitore werden in z.T. bereits in **ALOS** realisiert :

BEFOR-CREATE
AFTER-CREATE
BEFOR-DELETE
AFTER-DELETE Diese Methode wird getriggert, bevor bzw. nachdem eine Instanz der überwachten Klasse erzeugt bzw. gelöscht wird

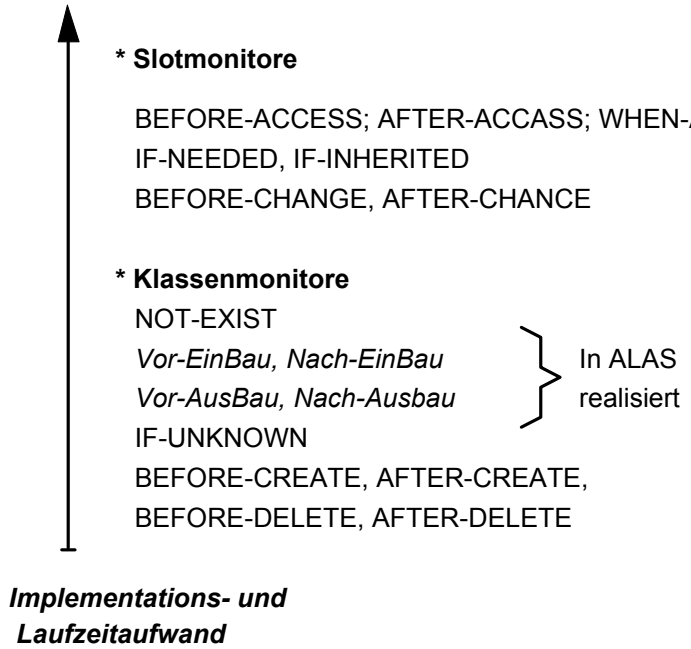
Vor-EinBau
Nach-EinBau
Vor-AusBau
Nach-AusBau Diese Methode wird getriggert, bevor bzw. nachdem eine Instanz der überwachten Klasse in eine Aufbaustruktur eingebaut bzw. ausgebaut wird

IF-UNKNOWN Diese Methode wird getriggert, wenn ein Zugriff zu einem unbekannten Slot erfolgt.

NOT-EXIST Monitor der Klasse EXTERN. Eine Methode die getriggert wird, wenn ein Objekt , das nicht existiert, eine Nachricht erhält

4.2.1. ALMS - AutoLisp Monitorsystem

Die Implementation der beschriebenen Monitore für Slots bereitet grundsätzlich keine größeren Schwierigkeiten. Da aber bei deren Verwendung jeder lesende bzw. schreibende Zugriff überwacht werden muß, führt dies zu einem Verlust an Performance. Der Einsatz von Monitoren lohnt daher nur, wenn statistisch gesehen viele Slots mit Monitoren versehen sind und wenn diese häufig feuern.
Die Klassenmonitore sind in dieser Hinsicht weniger problematisch, da das Erzeugen oder Löschen von Instanzen statistisch eher selten ist. Außerdem genügt hier eine Änderung der entsprechenden Funktionen von ALOS sowie das Hinzufügen weiterer standardisierter Methoden zur Klasse.
Es läßt sich folgende Reihenfolge des Implementations- und Ressourcenaufwandes aufstellen:



4.2.2. ALMS Referenz

(ErzeugeKlassenMonitor *classname monitor methodenkoerper*)
siehe ALOS - Modul '*AOMONITO.LSP*'.

4.3 Regeln

(Produktions-) Regeln besitzen eine wesentlich höhere Ausdruckskraft, da sie als if-then (-else) Konstrukte dargestellt werden, sich also einer Syntax bedienen, die entsprechenden umgangssprachlichen Formulierungen nahe kommt. Es gibt für sie verschiedene Standardauswertemechanismen, die im allgemeinen als Interpreter vorliegen. Eine Regelauswertung ist damit in jedem Fall resourcenintensiver als Monitorüberwachung eines Slots oder einer Klasse (zumindest, wenn ständig alle Regeln auf ihre Anwendbarkeit überprüft werden).
Beim einem Einsatz von Produktionsregeln in einer objektorientierten Modellierumgebung, können diese als *gerichtete* n:m Relationen über Instanzen oder Klassen verstanden werden.
Die Regel können über die am IF-Teil der Regel beteiligten Instanzen/Klassen diesen Objekten zugeordnet werden. Dies folgt dem Ansatz der OOM, daß alle einem Objekt zugehörigen Informationen auch in ihm gespeichert und damit zugreifbar sein sollen. In den meisten verfügbaren Systemen sind die Regeln jedoch selbständige Konstrukte der Domänenbeschreibung.

4.3.1. ALRES - AutoLisp Regelsystem

Datenstrukturen
Zur Nutzung des Regelsystemes wird ein globales Symbol `$AGEND` erzeugt, das die Liste noch auszuwertenden Regeln enthält. Das Symbol `$TRMOD` enthält ein Symbol das den aktuellen Protokollstatus für das Regeltracing enthält.

Nutzbare Funktionen

*(**ErzeugeRegel** rulename*
patternlist
predicate thenaction elseaction)
Erzeugen eines auswertbaren Regelausdrucks als Inhalt des Symboles *rulename* sowie Eintragen des *rulename* in den Slot 'RULES aller Klassen, die einem pattern aus *patternlist* genügen.
Parameter: *rulename* - Objektname Typ: SYMBOL
patternlist - Listen von Definition lokaler
Symbole und Hilfsvariablen folgender Struktur:
(*pattern ... / auxsymbol ...*)
Ein *pattern* besteht aus einer 2 elementigen Liste folgender Struktur:
(*classname lokalinstancename*)
predicte - S-Ausdruck, der nach Bindung der lokalen Symbole
ausgewertet wird und der T oder NIL zurück gibt
thenaction, elseaction - S-Ausdrücke, in denen entspr. der predicate-Auswertung die
lokalen Symbole gebunden und anschließen der Ausdruck
ausgewertet wird.
Rückgabe : T bei erfolgreicher Operation, sonst NIL

```
Beispiel    : (ErzeugeRegel     'R1_addiere
               '(  ('KOMPLEXFIGUREN inst) / tsum)
                        '(  (NOT (= (SETQ tsum (Summe inst 'flaeche))
                    (SendeNachricht inst 'flaeche)
                         ) )    )
                        '( (LokAendereAttribut inst 'flaeche tsum)
                          (Asserta inst)
                         )
                        '( )
          )
          T
```

*(**EvaluiereRegeln**)*
Die auf dem Regelstack (Symbol $AGEND) befindlichen Regeln werden gebunden und ausgewertet. Der Prozeß terminiert, wenn keine Regeln mehr auf dem Regelstack sind.
Parameter: Keine
Rückgabe : T bei erfolgreicher Operation, sonst NIL
Beispiel : (EvaluiereRegeln)
T

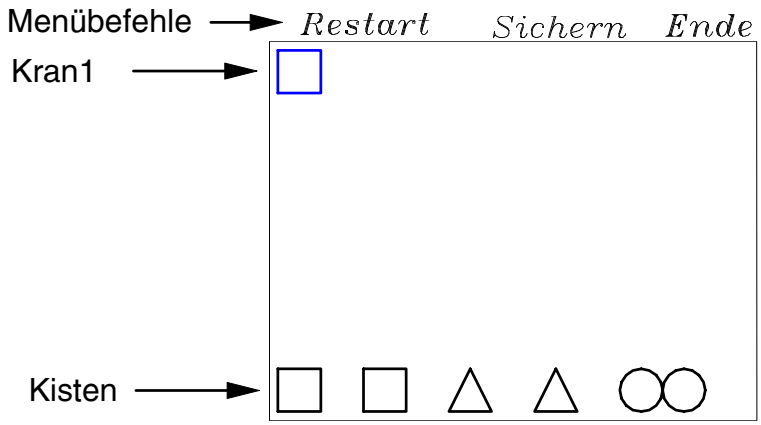
*(**Asserta** obj*)
Schreibt die Regeln auf den Rulestack, die im Slot 'RULES von *obj* stehen. (Nur wenn die Regel dort noch nicht steht)
Die Regel wird vorn angehängt, die Abarbeitung entspricht dadurch einer Tiefe-zuerst Strategie.
Parameter : *obj* - Objektname : Symbol
Rückgabe : T bei erfolgreicher Operation, sonst NIL
Beipsiel : (Asserta 'WC1)

(***Assertz** obj*)
Schreibt die Regeln auf den Rulestack, die im Slot 'RULES von *obj* stehen. (Nur wenn die Regel dort noch nicht steht). Die Regeln wird hinten angehängt, die Abarbeitung entspricht dadurch einer Breite-zuerst Strategie.
Parameter: *obj* - objektname : Symbol
Rückgabe : T bei erfolgreicher Operation, sonst NIL
Beispiel : (Assertz 'Duschen)

*(**ZeigeRegelProtokoll** level*)
Es wird der Wert des Symboles **$ARSTA** auf *level* gesetzt. Entsprechend des Levels werden verschieden detaillierte Protokolle gezeigt
'No - kein Protokoll 'Short - Kurzprotokoll
'Medium - Normalprotokoll 'Long - Langprotokoll
Das Umstellen des Protokollstatus ist auch während der Regelabarbeitung möglich, so das Regeln gezielt protokolliert werden können.
Parameter : level : neues Protokolllevel : SYMBOL
Rückgabe :alter Protokollstatus
Beispiel :(ZeigeRegelProtokoll 'SHORT)
LONG

5. Programmierbeispiel

Als Beispiel dient eine Kranwelt mit einem Kran sowie 6 Kisten (2 Kreise, 2 Dreiecke, 2 Vierecke als Instanzen der jeweiligen Klassen). Kisten können selektiert werden und werden auf den Stapel einer zu zeigenden Position gepackt. Die Zeichnung 'KRANWELT.DWG' ist zu laden, da sie die Blockdefinitionen enthält.




```
(LOAD "ALOS.LSP")     ; Laden des Objektsystms, alle Module werden geladen
(LOAD "MOV-SCAL.LSP") ; Funktione zum Bewegen und Skalieren von Blöcken
(LOAD "ATTRZUGR.LSP") ; Zugriff zu Blockattributen

;*************************************************************************
;******                                                            *****
;******    Kranwelt für AutoCad/AutoLisp  und Objektaufsatz ALOS     *****
;******    Version 1.0    1.11.92             (c) Frank Steinmann    *****
;******                                                            *****
;*************************************************************************
;
; -------- Erzeugen der Taxonomie ----------------------------------------
(ErzeugeStatischeKlasse  'Wurzel 'Extern)
(ErzeugeAttribut 'WURZEL 'x 0)
(ErzeugeAttribut 'WURZEL 'y 0)
(ErzeugeAttribut 'WURZEL 'ref NIL)

(ErzeugeStatischeKlasse 'Kraene     'Wurzel)
(ErzeugeAttribut 'Kraene 'Blkname "Kran")
(ErzeugeAttribut 'KRAENE 'kragrasp NIL)

(ErzeugeStatischeKlasse 'Rahmen     'Wurzel)
(ErzeugeAttribut 'Rahmen 'Blkname "Rahmen")

(ErzeugeStatischeKlasse   'Restartbutton 'Wurzel)
(ErzeugeAttribut 'Restartbutton 'Blkname "Restart")

(ErzeugeStatischeKlasse   'Sicherebutton 'Wurzel)
(ErzeugeAttribut 'Sicherebutton 'Blkname "Sichere")

(ErzeugeStatischeKlasse   'Endebutton    'Wurzel)
(ErzeugeAttribut 'Endebutton    'Blkname "Ende")

(ErzeugeStatischeKlasse 'Kranwelt   'Wurzel)
(ErzeugeAttribut 'KRANWELT 'ExemplarListe '())
(ErzeugeAttribut 'KRANWELT 'Heaps '((0)(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)))

(ErzeugeStatischeKlasse 'Kisten     'Wurzel)
(ErzeugeStatischeKlasse   'Viereck  'Kisten)
(ErzeugeAttribut 'Viereck  'Blkname "Viereck")

(ErzeugeStatischeKlasse   'Dreieck  'Kisten)
(ErzeugeAttribut 'Dreieck 'Blkname "Dreieck")

(ErzeugeStatischeKlasse   'Kreis    'Kisten)
```

```
(ErzeugeAttribut 'Kreis   'Blkname "Kreis")
;
; ------ Erzeugen der Klassenmethoden -----------------------------------
;
 (ErzeugeMethode 'Wurzel 'MakeExemplar '(name x y)
'( (ErzeugeStatischeInstanz name self "")
   (COMMAND "EINFÜGE" (Hole self 'Blkname) (LIST x y) 1 1 0)
   (LokAendereAttribut name 'x x)
   (LokAendereAttribut name 'y y)
   (LokAendereAttribut name 'ref (GetLastRef))
   (LokAendereAttribut 'kranwelt1 'Exemplarliste
              (CONS name (LokHole 'kranwelt1 'exemplarliste))   )
))
;.....................................................................
(ErzeugeMethode 'Kisten 'MakeExemplar '(name x y)
'( (ErzeugeStatischeInstanz name self "")
   (LokAendereAttribut name 'x x)
   (LokAendereAttribut name 'y y)
   (COMMAND "EINFÜGE" (Hole self 'Blkname) (LIST x y) 1 1 0)
   (LokAendereAttribut name 'ref (GetLastRef))
   (LokAendereAttribut 'kranwelt1 'Exemplarliste
              (CONS name (LokHole 'kranwelt1 'Exemplarliste))
   )
   (LokAendereAttribut 'kranwelt1 'Heaps
         (updprop (CONS name (getprop x (lokhole 'kranwelt1 'heaps)))
                  x (lokhole 'kranwelt1 'heaps)
)) )     )
;.....................................................................
(ErzeugeMethode 'Kranwelt 'MakeExemplar '(name)
'( (ErzeugeStatischeInstanz name self "")
))
;.....................................................................
(ErzeugeMethode 'Kraene 'MakeExemplar '(name x y)
'( (ErzeugeStatischeInstanz name self "")
   (LokAendereAttribut 'Kraene 'x x)
   (LokAendereAttribut 'Kraene 'y y)
   (COMMAND "EINFÜGE" (Hole self 'Blkname) (LIST x y) 1 1 0)
   (LokAendereAttribut name 'ref (GetLastRef))
   (LokAendereAttribut 'kranwelt1 'Exemplarliste
              (CONS name (LokHole 'kranwelt1 'exemplarliste))
   )
))
;.....................................................................
(ErzeugeMethode 'Kranwelt 'Kraninit '(name)
'(
;----- Initialisierung des Bildschirmes, von ACAD-Variablen
   (SETVAR "CMDECHO" 0)
   (COMMAND "LIMITEN" "-2,-2" "15,13")
   (COMMAND "ZOOM" "GRENZEN")(TERPRI)
   (COMMAND "BKSYMBOL" "AUS")
   (COMMAND "REFERENZ" "EIN")
   (SendeNachricht 'kranwelt 'MakeExemplar (List name))
;----- Bilden der Blockeinfügungen und initialisieren der entspr.
;----- Lispsymbole für die 6 Kisten, den Rahmen, den Kran sowie der Marken
;----- Initialisieren der lok.Attribute
(SendeNachricht 'dreieck 'MakeExemplar (LIST (GenSym) 4 0))
(SendeNachricht 'dreieck 'MakeExemplar (LIST (GenSym) 6 0))
   (SendeNachricht 'viereck 'MakeExemplar (LIST (GenSym) 0 0))
   (SendeNachricht 'viereck 'MakeExemplar (LIST (GenSym) 2 0))
   (SendeNachricht 'kreis   'MakeExemplar (LIST (GenSym) 8 0))
   (SendeNachricht 'Kreis   'MakeExemplar (LIST (GenSym) 9 0))
   (SendeNachricht 'restartbutton 'MakeExemplar '(restart -0.5 11.3))
   (SendeNachricht 'sicherebutton 'MakeExemplar '(sichere  4.5 11.3))
   (SendeNachricht 'endebutton    'MakeExemplar '(ende 8.5 11.3))
   (SendeNachricht 'Rahmen 'MakeExemplar '(ra 0 0))
   (SendeNachricht 'Kraene 'MakeExemplar '(kra 0 11))

;------ Neuzeichnen und Aufruf des Hauptmodules
   (COMMAND "NEUZEICH")
) )
;------- Ende 'Kraninit'

;=====================================================================
;                ***  Hauptmodul  ***
;  -  zur Schleifenbildung
;---------------------------------------------------------------------
(ErzeugeMethode 'kranwelt 'krane '(kranname / ki xki yki xziel yziel)
;----- Funkt. 'Welche' liefert Namen (Lispsymbol!) eines gepickten Blockes
;----- Schleife läuft bis gepickter Block 'ende ist
'( (WHILE (NOT (EQUAL (SETQ ki (SendeNachricht self 'Welche '() ))
                      'ende
```

```
          )    )
    (COND
;----- gepickter Block war Restartmarke
         ((EQUAL ki 'restart)
          (FOREACH Ele (Instanzen 'wurzel) (LoescheInstanz Ele))
          (COMMAND "LÖSCHEN" "k" "-20,-20" "100,100" "")
          (SendeNachricht 'kranwelt 'kraninit '(kranwelt1))
         )
;----- gepickter Block war Rahmen
         ((EQUAL ki 'rahmen)
          (PRINC "\nRahmen gepickt - noch mal !\n")
         )
;----- gepickter Block war Marke Sichern
         ((EQUAL ki 'sichere)
          (SichereObjektListe (Instanzen 'wurzel) "ALLINST")
          (PRINC "Objekte unter 'ALLINS.LOB' gesichert ! \n")
          (COMMAND "SICHERN" "")
          (PRINC  "Zeichenung unter akt. Zeichnungsnamen gesichert !\n")
         )
;----- es wurde eine Kiste gepickt
       (T
         (SETQ xki (LokHole ki 'x))            ; x-Koord. der Kiste
         (SETQ yki (LokHole ki 'y))            ; y-Koord. der Kiste
         (SETQ xziel (SendeNachricht self 'Spalte '() ))
                                               ; x-Koord. des Ziel picken
                                               ; mit Funkt. Spalte
;..... Kranfahrt
         (SETQ yziel (LENGTH (Getprop xziel (LokHole self 'heaps))) )
                                                    ; y-Koord. des Zieles
         (IF (= xki xziel) (SETQ yziel (1- yziel)))
         (SendeNachricht kranname 'hori (LIST xki))    ; fahren zur Kiste
         (SendeNachricht kranname 'verti (LIST yki))   ; senken zur Kiste
         (COMMAND "NEUZEICH")
         (SendeNachricht kranname 'fassen '())         ; fassen der Kiste
         (SendeNachricht kranname 'verti '(10))        ; heben der Kiste
         (COMMAND "NEUZEICH")
         (SendeNachricht kranname 'hori (LIST xziel))  ; fahren mit Kiste
         (SendeNachricht kranname 'verti (LIST yziel)) ; senken mit Kiste
         (COMMAND "NEUZEICH")
         (SendeNachricht kranname 'lassen '())         ; lassen der Kiste
         (SendeNachricht kranname 'verti '(10))        ; heben in Grundstellung
         (COMMAND "NEUZEICH")                          ; Zeichnung regenerieren
     )  )
   ) ;..... EndWhile
;----- gepickter Block war Endemarke
   (COMMAND "LÖSCHEN" "k" "-20,-20" "100,100" "")    ; Löschen des Bildschirms
;   (LoescheObjekt 'wurzel)                          ; Löschen aller Objekte
   (PRINC "\n ----  Tschüß ! -----\n") T
))
```

```
;.....................................................................
(ErzeugeMethode 'kranwelt 'Spalte '()  '((Spalte))     )
;.....................................................................
; Picken eines Kranweltelementes und Rückgabe dessen Blknamens (Symbol)
(ErzeugeMethode 'Kranwelt 'welche '(/ pickref templi)
'(     (SETQ templi (LokHole 'kranwelt1 'ExemplarListe))
;----- Picken eines Zeichnungselementes und Rückgabe seines ACAD-Datenbankpoint.
       (SETQ pickref (GetRef (Pickblock "Kiste oder Marke picken!  ")))
;----- Suchen des Namens (Lispsymbols) des gepickten Zeichnungselementes
       (WHILE (AND templi
                   (NOT (EQUAL pickref (LokHole (CAR templi) 'ref) ))
              )
              (SETQ templi (CDR templi))
       )
;----- Erstellen des Rückgabewertes oder Fehler
       (IF    (AND templi (EQUAL pickref (LokHole (CAR templi) 'ref )))
              (CAR templi)
              (ObjErr self 'Welche_Keine_objekt templi)
 )     )
)

;---------------------------------------------------------------------------
; *****  benuzte Methoden  *****
;---------------------------------------------------------------------------
; Horizontales Fahren des Kranes mit oder ohne Kiste
;
(DEFUN hori (x / i dx kiname Krapoi Kipoi xkra)
    (SETQ xkra (LokHole 'kra 'x))             ; x-pos. des Kranes
    (SETQ i  (ABS (- x xkra)))                ; x-Verschiebung in Einerschritten
    (SETQ Krapoi (HANDENT (LokHole 'kra 'ref)))  ; ACAD-Datenbankrefnter zur
                                                 ; Blockeinfügung des Kranes
    (IF (/= i 0) (PROGN
       (SETQ dx (/ i (- x xkra)) )            ; dx immer 1 oder -1
       (SETQ kiname (LokHole 'kra 'kragrasp)) ; Kiste am Kran? Name oder NIL
       (IF kiname
;----- fahren mit Kiste
           (PROGN (SETQ Kipoi (HANDENT (LokHole kiname 'ref)))
                  (REPEAT i (Move Krapoi dx 0)         ; verschieben des Kranes
                            (Move  Kipoi dx 0)    )    ; verschieben der Kiste
                  (LokAendereAttribut 'kra 'x x)       ; updaten der Krankoord.
                  (LokAendereAttribut  kiname 'x x)    ; " der Kist.-koor
           )
;----- fahren ohne Kiste
           (PROGN (REPEAT i (Move Krapoi dx 0))        ; verschieben des Kranes
                  (LokAendereAttribut 'kra 'x x)       ; updaten der Krankoord.
)   )   )  ) T   )
;.....................................................................
; Vertikales Fahren des Kranes mit oder ohne Kiste
;
(DEFUN verti (y / kiname)
   (SETQ kiname (LokHole 'kra 'kragrasp ))              ; Kiste gefasst?
   (IF kiname
;------  heben/senken mit Kiste
       (PROGN (Move (HANDENT (LokHole kiname 'ref))     ; verschieben der Kiste
                    0 (- y (LokHole kiname 'y))
              )
              (LokAendereAttribut kiname 'y y)          ; Updaten der
   )   )                                                ; Kistenkoordinaten
;-----
   (Verz (HANDENT (LokHole 'kra 'ref)) 1  (+ (- 10 y) 1)   )
                                                        ; Verzerren des Kranes
   (LokAendereAttribut 'kra 'y y)                       ; updaten der Krankoor.
)
```

```
;.....................................................................
; Fasseoperation (updaten der Datenbasis)  Rückgabe : T
;
(DEFUN fassen (/ x heaps heap)
   (SETQ x (LokHole 'kra 'x) )                    ; x-pos des Kranes
   (SETQ heaps (LokHole 'kranwelt1 'heaps))       ; alle Kistenstapel
   (SETQ heap (Getprop x heaps ) )                ; Kistenstapel in x-pos
   (LokAendereAttribut 'kra 'kragrasp (CAR heap) )  ; Upd. der Fasse-attr.
                                                  ; des Kranes
   (LokAendereAttribut 'kranwelt1 'heaps (Updprop (CDR heap) x heaps))
                                                  ; Upd. der Stapelliste
 T                                                ; d.h. (reduzieren) des Kiste-
)                                                 ; stapels in xpos
;.....................................................................
; Lasseoperation (updaten der Datenbasis)  Rückgabe : T
(DEFUN lassen (/ x heaps heap)
   (SETQ x (LokHole 'kra 'x))              ; x-pos des Kranes
   (SETQ heaps (LokHole 'kranwelt1 'heaps)) ; Kistenstapel in x-pos
   (SETQ heap (Getprop x heaps))
   (LokAendereAttribut 'kranwelt1 'heaps
                   (Updprop (CONS (LokHole 'kra 'kragrasp) heap) x heaps)
   )                                      ; Upd. der Stapelliste
   (LokAendereAttribut 'kra 'kragrasp 'NIL)  ; d.h. (erhöhen) des Kiste-
   T                                      ; stapels in xpos
)
;.....................................................................
; Picken einer Spaltenposition
; Return integer (0..10)
(DEFUN Spalte ( / temp)
 (SETQ temp (FIX (+ 0.5 (CAR (GETPOINT "Ziel positionieren !\n")))))
 (COND ((< temp 0)    0)
       ((> temp 10)  10)
       (T          temp)
))
;=====================================================================
; ***** Erzeugen von Methoden aus Funktionen *****
;=====================================================================
(ErzeugeMethode 'kraene 'hori '(x) '((hori x)))
;.....................................................................
(ErzeugeMethode 'kraene 'verti  '(y) '((verti y)))
;.....................................................................
(ErzeugeMethode 'kraene 'fassen '() '( (fassen) )
)
;.....................................................................
(ErzeugeMethode 'kraene 'lassen '() '( (lassen)  )
)

; *********************************************************************
; *****     benutzen der Kranwelt                                 *****
; *********************************************************************
 (SendeNachricht 'Kranwelt 'kraninit '(kranwelt1))

 (SendeNachricht 'kranwelt1 'krane '(kra))
```

ANHANG F Laufzeitvergleich ALOS – KappaPC (Aug. '96)

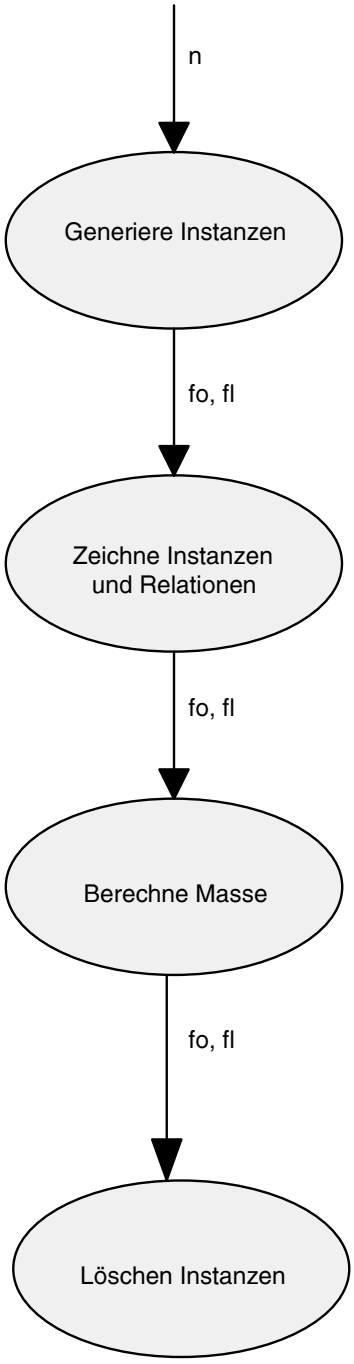
Autor Dipl.-Ing. F. Steinmann

Testumgebung : Pentium 100 MHz, 24 MB RAM

Testproblem : Dynamisches Instanziieren von 7 Klassen von 2D-Objekten

Funktionales Modell

n : Anzahl von Figuren
fo : erstes Objekt der Relationenkette
fl : letztes Objekt der Relationenkette



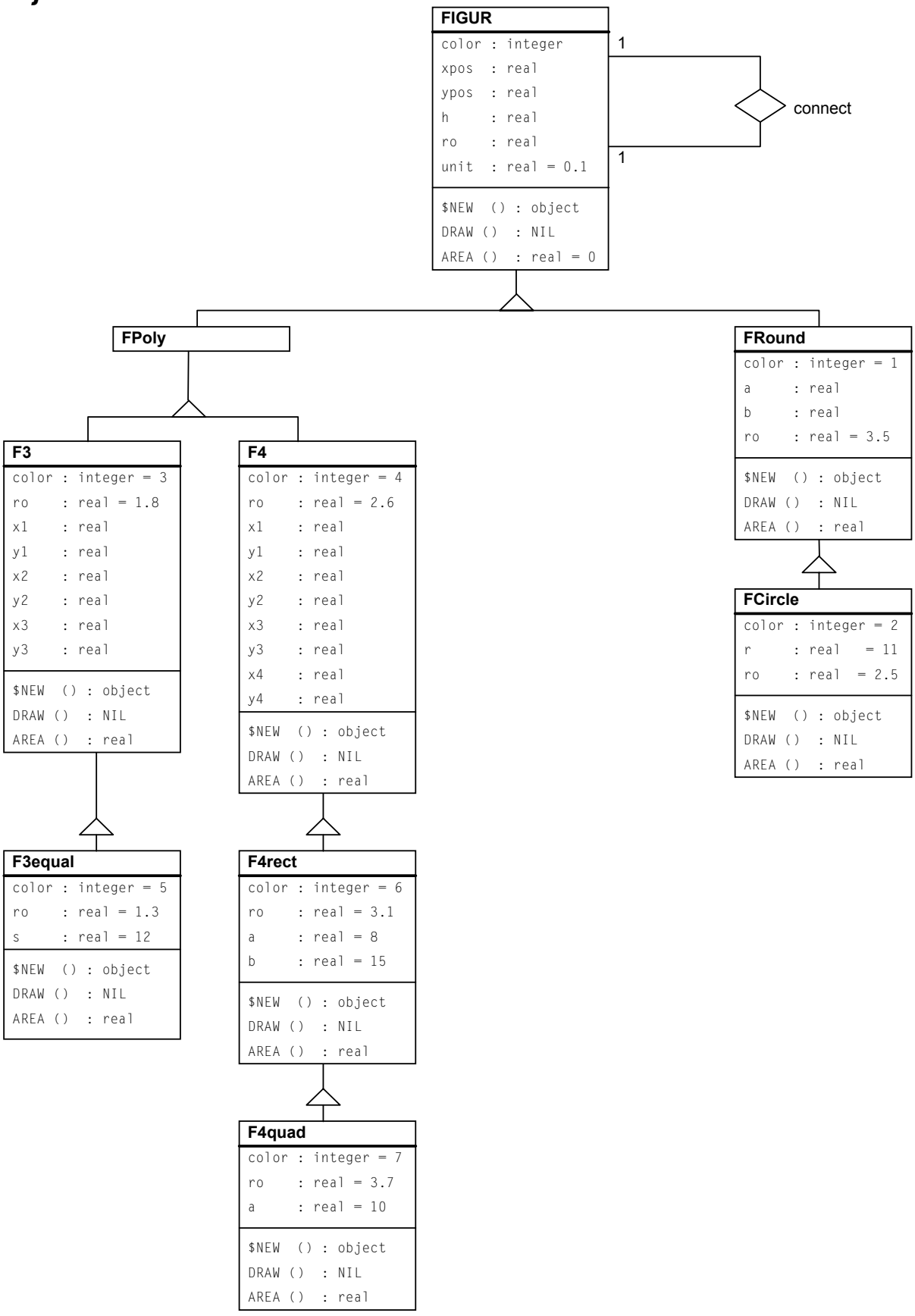
- Generiere n Instanzen zufällig aus den Klassen FRound, FCircle, F3, F4 F3equal, F4rect, F4quad
- Setze connect Relation für aufeinanderfolgend generierte Objekte

- Zeichne Instanzen
- Zeichne Linie für Objekte mit connect Relation

- Berechne Masse aller Instanzen entlang der connect Relation mit Formel M = area * h * ro
- Kummuliere Masse aller Instanzen entlang der connect Relation mit geradzahligen Farbindex

- Lösche alle Instanzen entlang der connect Relation

Objektmodell

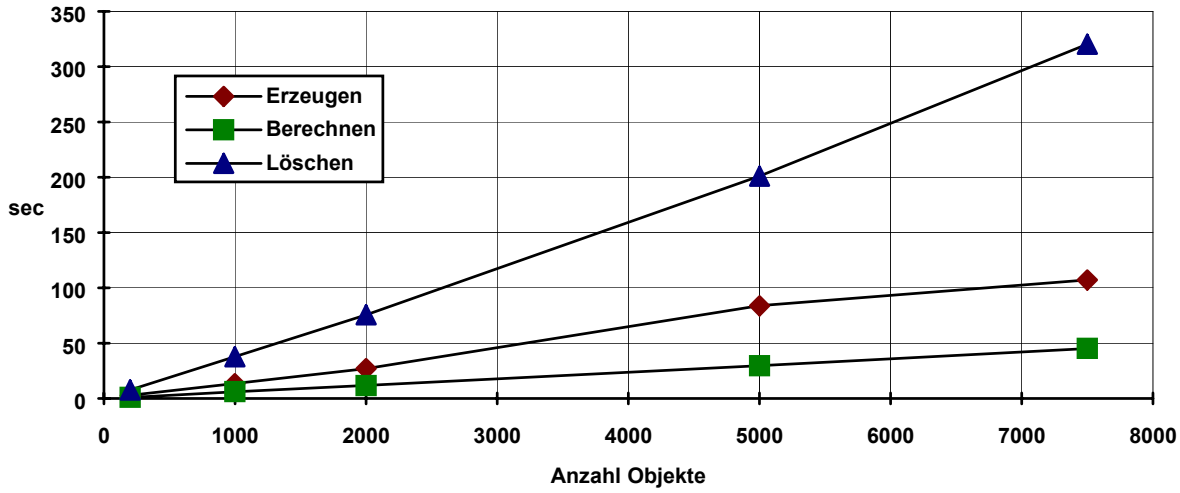


Laufzeittest ALOS

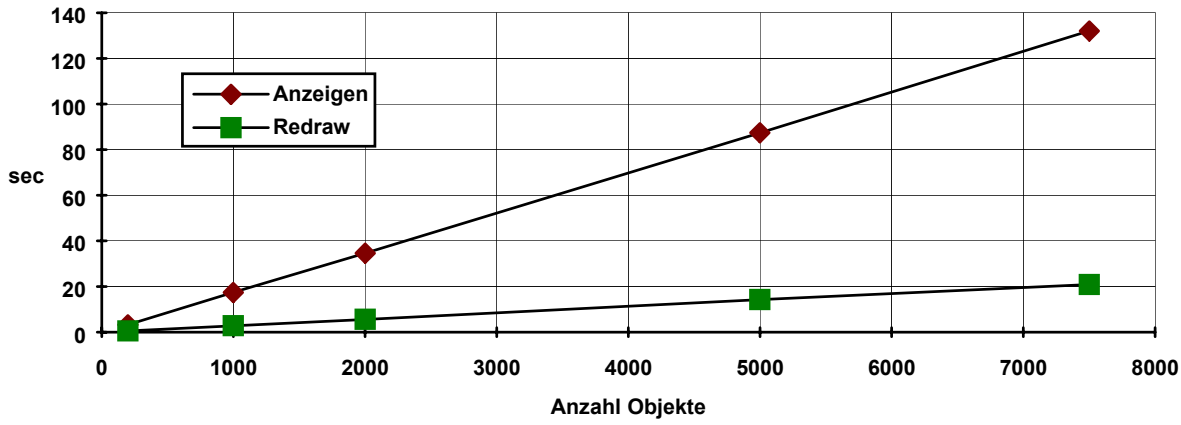
Laufzeitdaten

Anzahl	*50*	*100*	*200*	*500*	*800*	*1000*	*1600*	*2000*	*5000*	*7500*
Erzeugen	0.66	1.53	2.8	7.14	11.43	13.51	23.13	26.91	83.93	107.16
Anzeigen	1.1	1.54	3.3	8.51	13.67	17.47	27.07	34.66	87.39	132.15
Berechnung	0.33	0.44	0.93	2.92	4.4	6.15	9.07	11.7	29.6	45.21
Löschen	1.92	3.85	7.64	19.05	30.59	37.85	59.76	75.58	200.97	320.32
REDRAW	0.22	0.27	0.55	1.43	2.2	2.8	4.55	5.6	14.28	20.93

Zeichnungsunabhängige Funktionen



Zeichnungsabhängige Funktionen

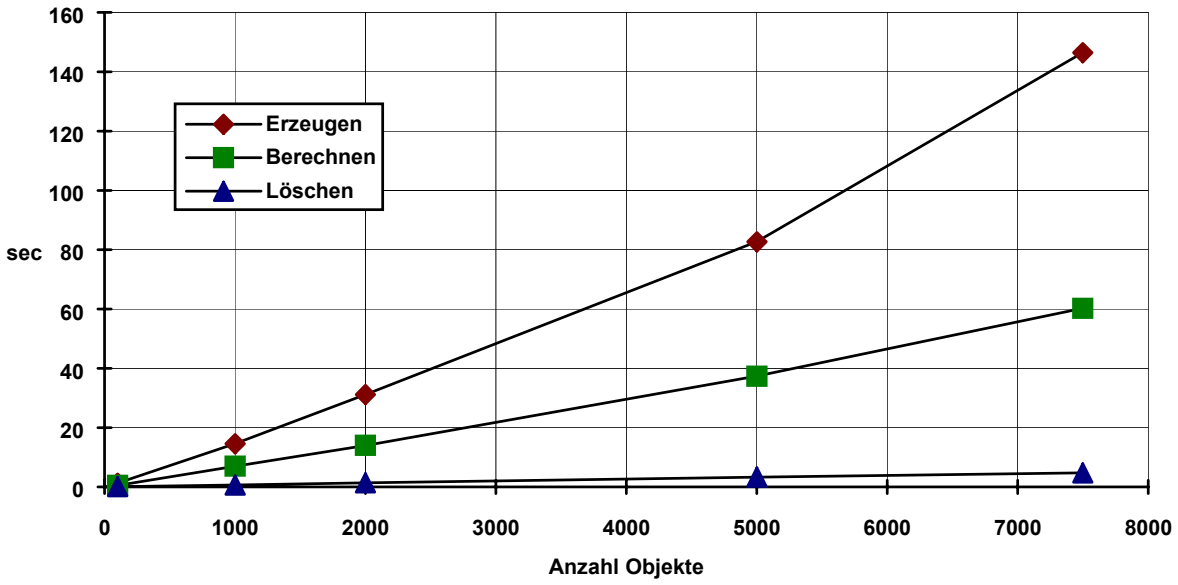


Laufzeittest KappaPC

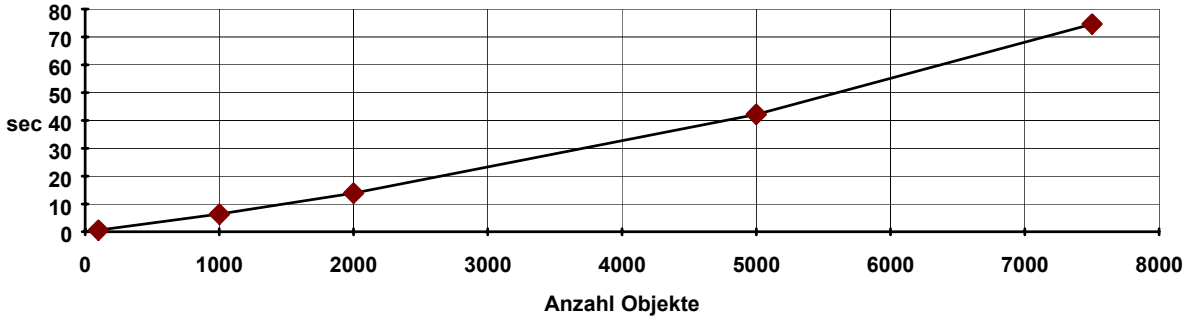
Laufzeitdaten

Anzahl	50	100	200	500	800	1000	1600	2000	5000	7500
Erzeugen	0.6	1.3	2.6	6.6	10.9	14.6	23.7	31.2	82.7	146.5
Anzeigen	0.3	0.6	1.2	3.1	5.2	6.4	10.6	13.9	42.2	74.6
Berechnung	0.3	0.5	1.3	3.0	5.1	7.0	11.0	14.0	37.4	60.3
Löschen	0.1	0.1	0.2	0.3	0.5	0.7	1.0	1.4	3.3	4.8
REDRAW										

Zeichnungsunabhängige Funktionen

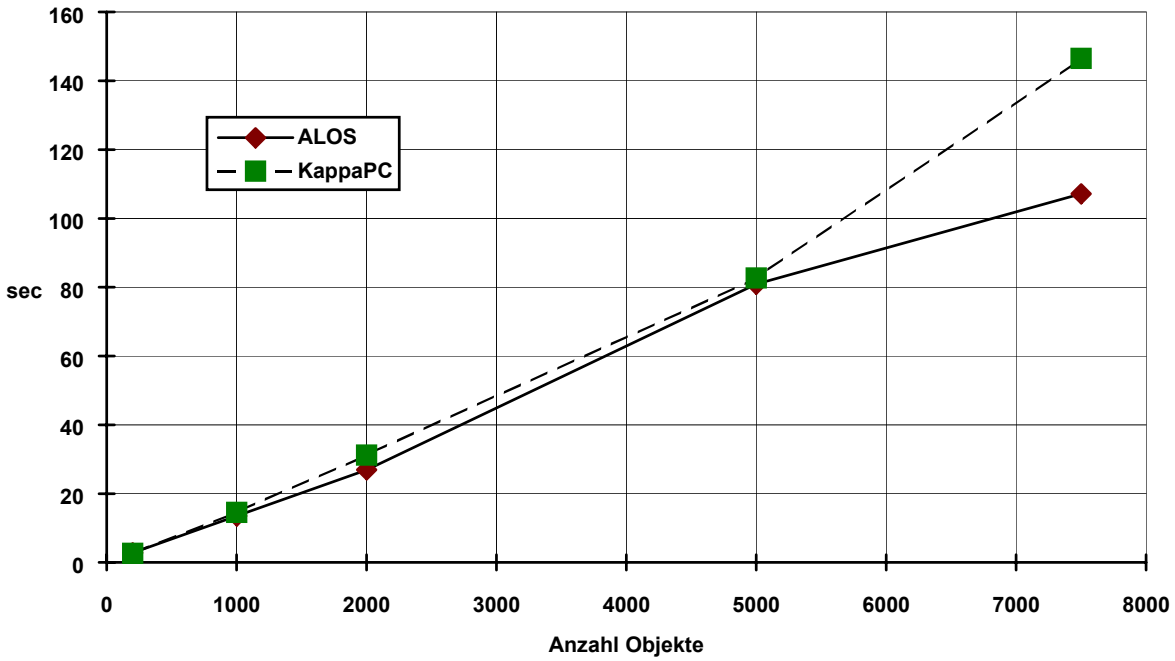


Zeichnen

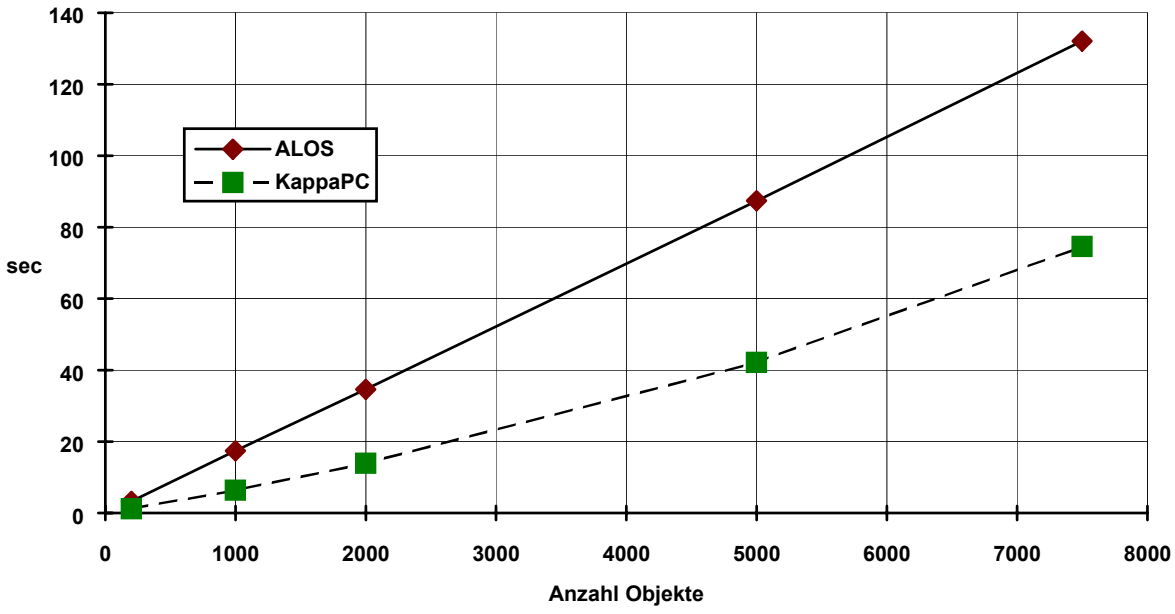


Vergleich der Objektsysteme

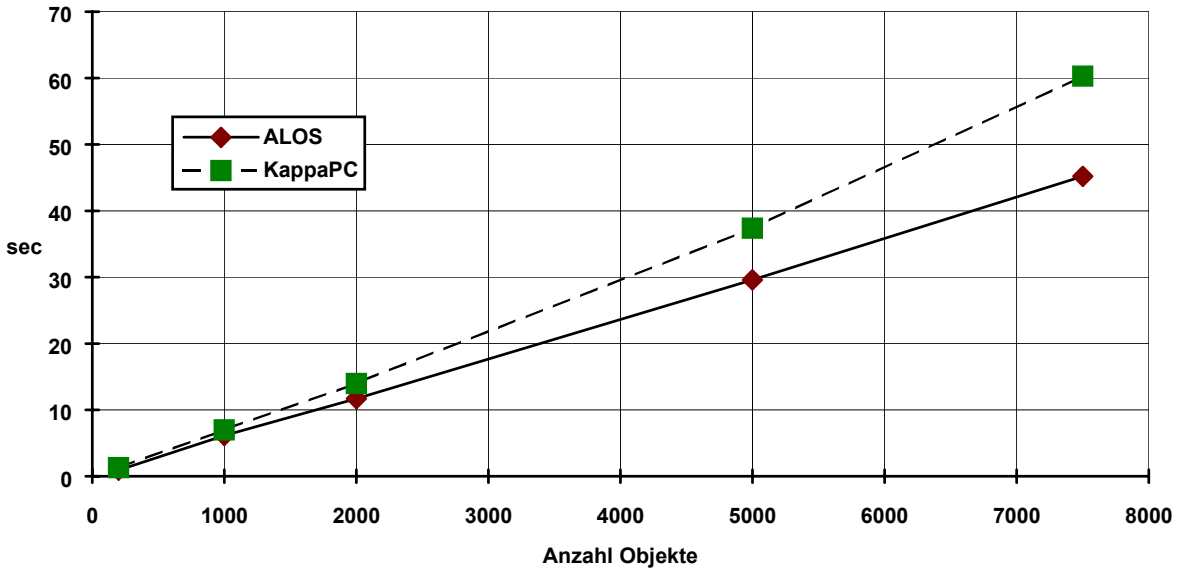
Erzeugen



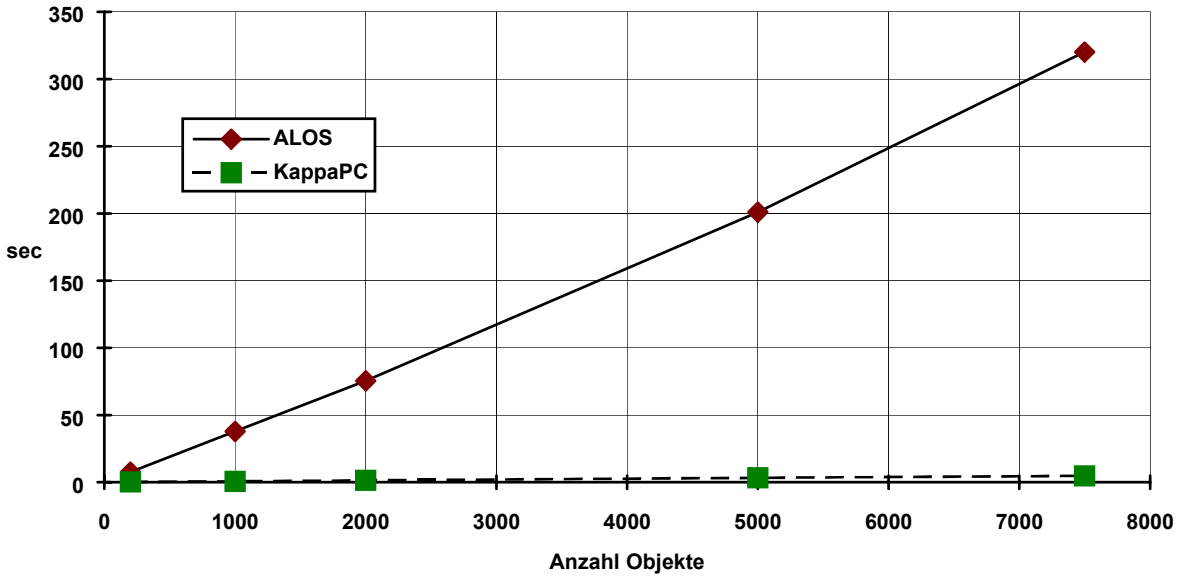
Anzeigen



Berechnen

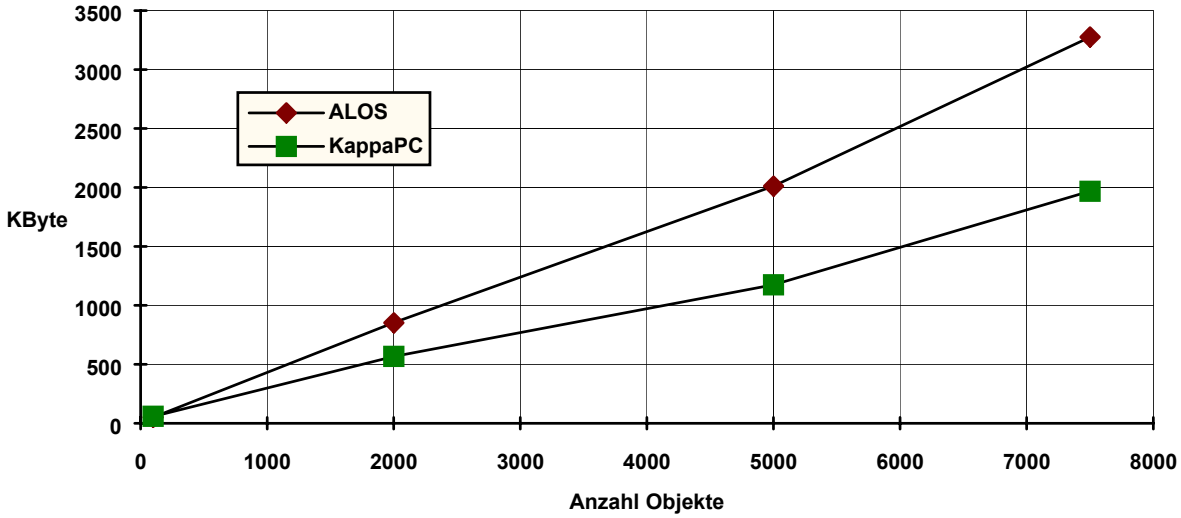


Löschen



Speicherbedarf der Objekte

	100	***2000***	***5000***	***7500***
ALOS	52	854	2012	3275
KappaPC	58	567	1175	1969



Bemerkung : ALOS verfügt gegenüber KappaPC über Objektidentität, d.h. es wird zwischen Objekten und deren Name unterschieden. Somit muß ALOS entsprechende Tabellen zur Verwaltung uniker Objektidentifikatoren und Objektnamen führen. Dies läßt sich allerdings abstellen und verbessert die Laufzeit dann signifikant.

KappaPC brach bei der Erzeugung von mehr als 7500 Objekten mit Fehler ab.

ALOS wurde bis 12500 Instanzen der beschriebenen Klassen getestet.

Bei beiden Systemen (jedoch besonders bei ALOS) differierten Speicherbedarf und Laufzeiten in Abhängigkeit davon, zum wievielten mal ein Testprogramm innerhalb einer Session der jeweiligen Entwicklungsumgebung gestartet wurde.

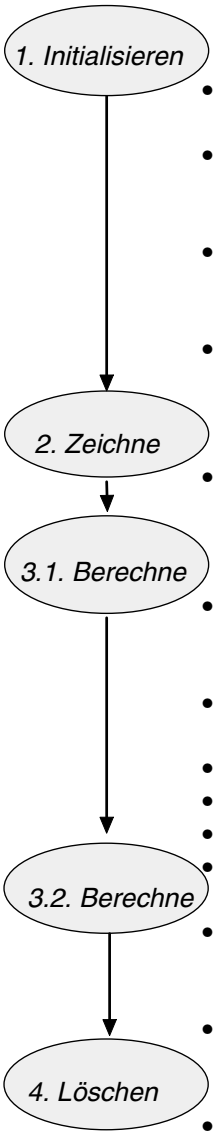
Anhang G - Laufzeittest FLEXOB (März '97)

Autoren : Dipl.-Ing. F. Steinmann
Dipl.-Inf. R. Wehner

Testumgebung : Pentium 100 MHz, 64 MB RAM, Windows NT 3.51

Testproblem : Dynamisches Instanziieren von 10 Klassen von Objekten sowie deren dynamische Attributierung, Zeichnen und Berechnungen der Objektmenge

Funktionales Modell



- Es gibt eine Attributobjektklasse 'Wert', abgeleitet von 'SingelNumber'
- Alle Raumobjekte und deren Subklassen verfügen über eine Methode 'Berechne_Gesamtpreis', die für ein Objekt seinen Preis (vom Typ 'Wert') addiert mit dem Preis aller seiner 'Part-Of'-Objekte sowie allen weiteren Attributen vom Typ 'Wert'.

- erzeuge ***n*** Räume, wobei Instanzen aus allen Klassen (8) mit gleicher Häufigkeit erzeugt werden.
- initialisiere zufällig (aus dem angegebenen Bereich) die Attribute (Value-Facette) X,Y, Breite, Laenge und Preis, trage laufende Nummer der Erzeugung als Namensstring ein
- die 'verbunden' Relation wird als sequenzielle Liste gebildet (eigentlich 1:n Relation wird aber nur 1:1) benutzt), wobei die fortlaufend erzeugten Objekte als jeweils miteinander verbunden eingetragen werden
- erzeuge für jede erzeugte Rauminstanz (oder deren Subklassen) ein Inventarobjekt als dessen Teil (Has-Part), initialisieren in ihm zufällige Preis, Inventarnummer und setzte X,Y auf die Werte seines 'Part_Of' - Objektes

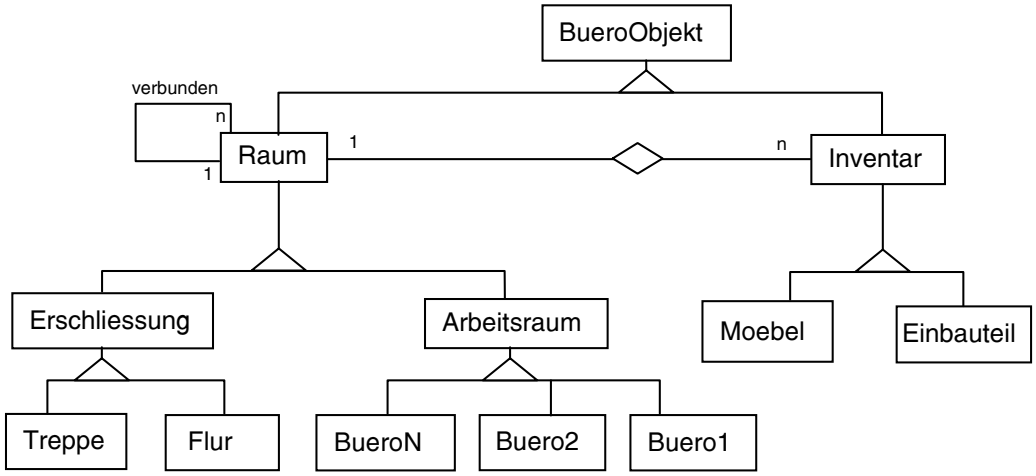
- Zeichne alle Objekte (Räume und Inventar) entlang der 'verbunden' sowie der 'Part-of' Relation (da sequenzielle Liste also alle) mit ihrem Farbindex und mit ihrer jeweiligen Zeichenmethode

- Ermittle ausgehend von einem gegebenen Raum (dem erster erzeugten Raum), den durchschnittlichen VolumensPreis aller erreichbaren Räume (da sequenzielle Liste hier alle) wie folgt :
- Berechne für jedes Raumobjekt den GesamtPreis GP aus eigenem Preis + Preis der Inventarobjekte die 'Part-Of' sind
- summiere alle GesamtPreise der Räume in GGP
- ermittle das Volumen V eines Raumes aus eigener Methode 'Area' * Hoehe
- summiere alle Volumen in GV
- ermittle Durchschnitt als GGV / GV

- Erzeuge dynamisch bei der Klasse 'Raum' zwei Attribute 'Fensterpreis' und 'Tuerpreis' vom Typ 'Wert' und initialisiere deren Value-Facette mit Zufallswerten bei *allen* Instanzen von Raum (und dessen Subklassen)
- berechne wie 3.1. wobei die Preisermittlung jedoch nun über die Methode 'Berechne_Gesamtpreis' gehen soll !

- Lösche alle Raum- und InventarObjekte, ausgehend von einem gegebenen RaumObjekt (erster erzeugter Raum), alle erreichbaren Räume (also alle)

Objektmodell

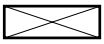


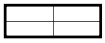
BueroObjekte		
Name	string	„BüroObjekt“
Farbe	integer	123123123
X	float	0 (0 ... 100)
Y	float	0 (0 ... 100)
Preis	float	1000 (0 ... 1000000)
Draw ()		Kreuz (01.*0.1) mit Frabe an (X,Y)

Flur		
Name	string	„Erschließung“
Funktion	string	„horizontal erschließen“
Farbe	integer	321321321

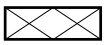
Treppe		
Name	string	„Treppe“
Funktion	string	„vertikal erschließen“
Farbe	integer	456456456
Hoehe	float	3 (0 ... 1)
Draw ()		Zeichnet Rechteck mit 5 horizontalen Linien an (X,Y)
Area ()	float	Breite*Höhe / 2

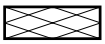
Raum		
Name	string	„Raum“
Funktion	string	„nicht spezifiziert“
Farbe	integer	234234234
verbunden	Raum*	()
Hoehe	float	1 (0 ... 10)
Breite	float	5 (0 ... 10)
Laenge	float	5 (0 ... 10)
Berechne_ Gesamt-preis’ ()	float	
Draw ()		Zeichnet Rechteck ungefüllt mit Farbe an (X,Y)
Area()		Breite*Laenge

Arbeitsraum		
Name	string	„Arbeitsraum“
Funktion	string	„arbeiten“
Farbe	integer	987987987
		Zeichnet Rechteck mit liegendem Kreuz in Farbe an (X,Y)

Erschliessung		
Name	string	„Erschließung“
Funktion	string	„erschließen“
Farbe	integer	321321321
Draw ()		Zeichnet Rechteck mit stehendem Kreuz mit Farbe an (X,Y)

Buero1		
Name	string	„Einmannbüro“
Farbe	integer	123456789
Breite	float	2 (0 ... 10)
Laenge	float	2 (0 ... 10)
Inhaber	string	„unbekannt“

Buero2		
Name	string	„Zweimannbüro“
Funktion	string	„arbeiten“
Farbe	integer	386672415
Breite	float	4 (0 ... 10)
Laenge	float	2 (0 ... 10)
Draw ()		Zeichnet Rechteck mit zwei liegenden Kreuzen

BueroN		
Name	string	„Großbüro“
Funktion	string	„arbeiten“
Farbe	integer	403541543
Breite	float	6 (0 ... 10)
Laenge	float	3 (0 ... 10)
Draw ()		Zeichnet Rechteck mit vierliegenden Kreuzen

Einbauteile		
Name	string	„Festinstallationen“
Farbe	integer	666333222
Seite	float	0.1 (0 ... 10)
Draw ()		Zeichnet Quadrat mit Seitelänge=Seite bei (X,Y) in Farbe mit Farbe gefüllt

Inventar		
Name	string	„Inventar“
Funktion	string	„nicht spezifiziert“
Farbe	integer	333555111
Inventarnummer	integer	0

Moebel		
Name	string	„Möbel“
Farbe	integer	3817575678
Radius	float	0.1 (0 ... 10)
Draw ()		Zeichnet Kreis mit Radius bei (X,Y) in Farbe mit Farbe gefüllt

Ergebnisse

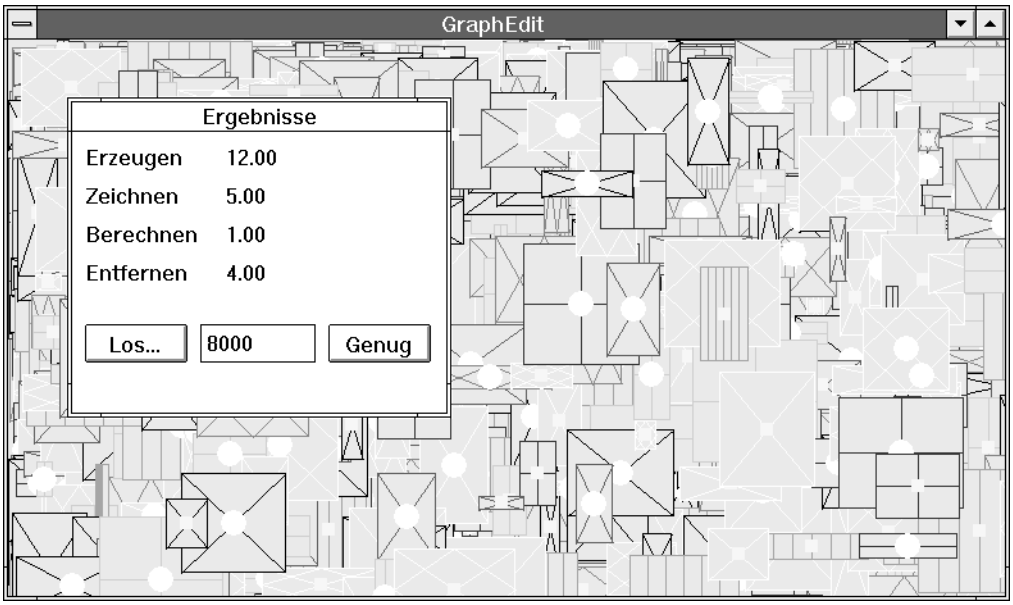
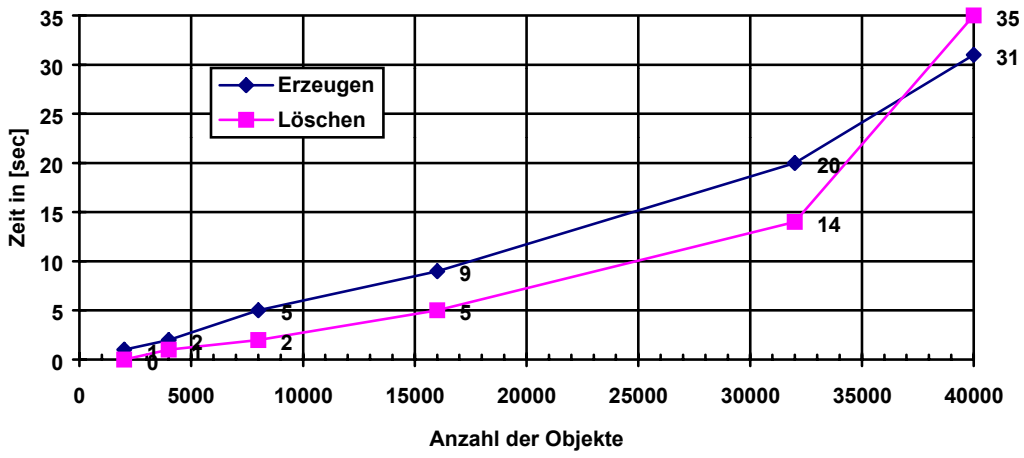


Abb. G-1 HardCopy Testprogramm (10 Objekte)

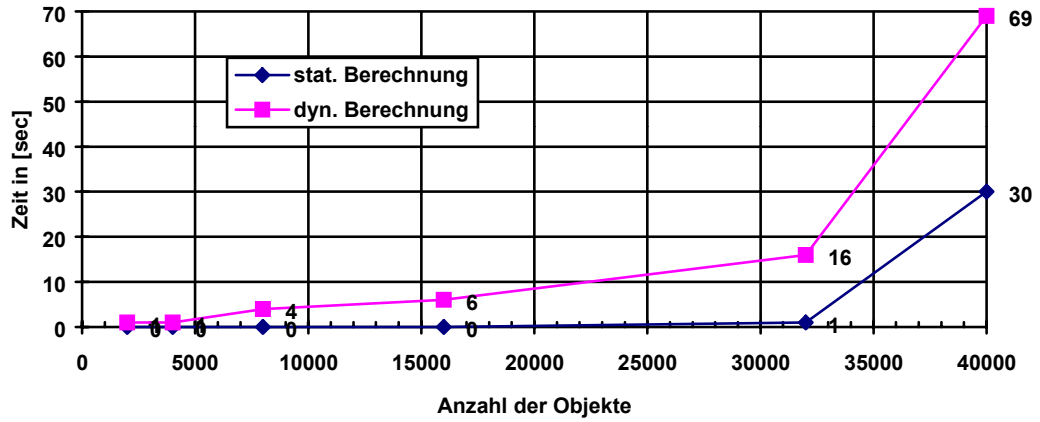
Laufzeiten der einzelnen Tests (in [sec])

Anzahl	***2000***	***4000***	***8000***	***16000***	***32000***	***40000***
Erzeugen	1	2	5	9	20	31
Anzeigen	1	2	3	6	18	61
Berechnung 3.1	0	0	0	0	1	30
Berechnung 3.2	1	1	4	6	16	69
Löschen	0	1	2	5	14	35

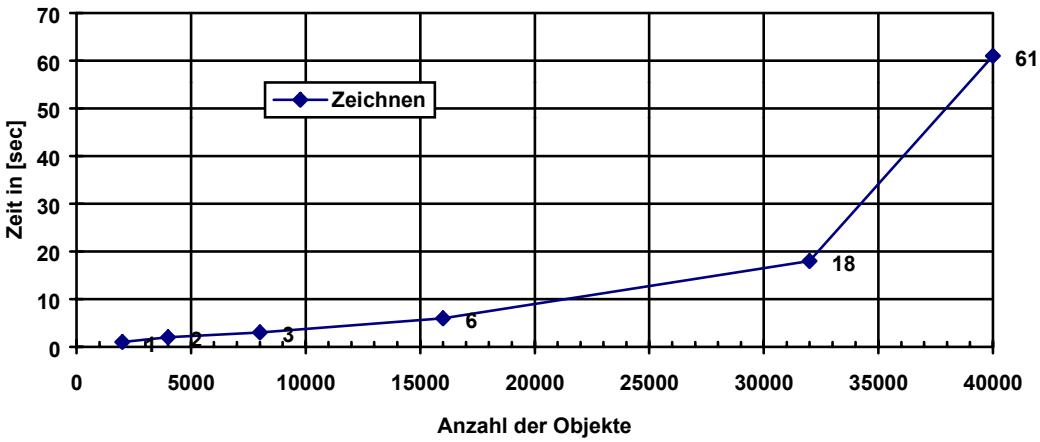
Funktionen zum Erzeugen/ Löschen von Objekte



Berechnungsfunktionen



Zeichnungsfunktionen



Bemerkung: Für die Ermittlung des Speicherbedarfs der Objekte wäre ein sehr hoher Aufwand zu treiben. Bei 60'000 Objekten setzt in der verwendeten Rechnerkonfiguration mit 64 MByte Hauptspeicher Swaping durch das Betriebssystems ein. Der Speicherbedarf differiert bedingt durch den Suballocator von Borland C++ sowie durch das Speichermanagement von Windows NT.

Lebenslauf

Name:	Frank Steinmann, seit 8 Jahren in Lebensgemeinschaft mit Heike Klenner
Kinder:	Max (8) und Julius Klenner (6)
Anschrift :	99427 WEIMAR, Berliner Str.32
Vater:	Roland Steinmann - Lehrer
Mutter:	Edda Steinmann, geb. Beinicke - Lehrerin

Tabellarischer Lebenslauf

10. Juni 1962	geboren in Sömmerda (Thüringen)
1969 - 1977	Besuch der Polytechnischen Oberschule Kindelbrück (Landkreis Sömmerda)
1977 - 1981	Besuch der Erweiterten Oberschule Buttstädt
1981	Abitur (Prädikat : sehr gut)
1981 - 1983	18 Monate Grundwehrdienst
1983 - 1988	Aufnahme des Studiums des „Bauingenieurwesens“, Fachrichtung „Informationsverarbeitung im Bauwesen“, an der Hochschule für Architektur und Bauwesen Weimar
1986	Praktikum am Rechenzentrum der Technischen Universität Dresden
1988	Diplomabschluß, Prädikat: sehr gut, Gesamtabschluß : gut
1988	Forschungsstudiums am Bereich Softwaretechnik der Hochschule für Architektur und Bauwesen Weimar
1991	Wissenschaftlicher Assistent am Bereich Informations- und Wissensverarbeitung der Hochschule für Architektur und Bauwesen Weimar

Weimar, den 4.5.1997 Unterschrift

wissenschaftlicher Werdegang

Name: Dipl.-Ing. Frank Steinmann

Anschrift: 99427 WEIMAR, Berliner Str.32

Tätigkeit: Wissenschaftlicher Assistent

Bereich Informations- und Wissensverarbeitung / Prof.Dr. R. Hübler
Fakultät Bauingenieurwesen
Bauhaus-Universität Weimar

1981	• Abitur
1983	• Aufnahme des Studiums des „Bauingenieurwesens“, Fachrichtung „Informationsverarbeitung im Bauwesen“, an der Hochschule für Architektur und Bauwesen Weimar
1986	• Praktikum am Rechenzentrum der Technischen Universität Dresden, Thema: „Testwerkzeuge zur Entwicklung von Grafikprogrammen“
1988	• Diplom, Thema : "Untersuchung zu einem Experimental Experten-system ’Entwurf von Trägerlagen’ "
1988	• Aufnahme des Forschungsstudiums am Bereich Softwaretechnik der Hochschule für Architektur und Bauwesen Weimar
1990	• Sprachkundigenprüfung 2a (Englisch)
1991	• Anstellung als wissenschaftlicher Assistent am Bereich Informations- und Wissensverarbeitung der Hochschule für Architektur und Bauwesen Weimar
1991 - 1997	• Betreuung von 12 Diplomarbeiten • Selbständige Konzeption und Durchführung von Lehrveranstaltungen zu den Themen „Neuronale Netze“ und „Lisp“ • Mehrere Arbeitsberichte und wiss. Projektvorschläge • Mehrere internationale Veröffentlichungen und Vorträgen • Realisierung des experimentellen Objektorientierten Modellverwaltungssystems ALOS

- Bearbeitung und inhaltliche Gestaltung der Projektbearbeitung „intelligente CAAD-Systeme“ im Rahmen des DFG-Forschungsschwerpunktes ‘Objektorientierte Modellierung in Planung und Konstruktion’[1]
- Mehrere eingeladene Vorträge in Industrie und an Forschungseinrichtungen
- Initiierung und Koordinierung des ‘Arbeitskreises Objekte’, der Bauhaus-Universität Weimar, Techn. Universität Dresden, Techn. Universität Berlin, Hochtief Software GmbH zur Konzeption und Realisierung einer Metaobjektschnittstelle

Veröffentlichungen

Hübler, R. ; Steinmann, F. : "Wissensbasierter Entwurf von Trägerlagen - ein Beitrag zur Automatisierung von Syntheseaufgaben",
Wiss. Zeitschrift der HAB Nr. 36 (1990), Weimar, 1990

Steinmann, F. : "Wissensverarbeitung- alter Hut mit neuen Federn ?",
Mikroprozessortechnik 2/91, Verlag für Technik, Berlin 1991

Hübler, R. ; Steinmann, F. : „Beitrag zur Computerstützung früher Phasen des architektonischen Entwurfes", in Tagungsband 14. IKM, HAB Weimar, Weimar 1994

Steinmann, F. :"Die Anwendung des Prinzips der sensorischen Karten zur Visualisierung komplexer Beziehungsnetze", in Tagungsband 14. IKM, HAB Weimar, Weimar 1994

Hübler, R.; Kolbe, P. ; Steinmann, F. : „Wissensbasierte Computerstützung früher Phasen des architektonischen Entwurfs, Teil I - Konzeption und Realisierung des Systems PREPLAN“, in „Computer und Architektur - Computereinsatz in frühen Entwurfsphasen“, in Wissenschaftliche Zeitschrift der HAB Weimar, Heft 4/94, Weimar, 1994

Steinmann, F. : „Wissensbasierte Computerstützung früher Phasen des architektonischen Entwurfs, Teil II – Wissensorganisation und Unschärfe im System PREPLAN“, in „Computer und Architektur - Computereinsatz in frühen Entwurfsphasen“, Wissenschaftliche Zeitschrift der HAB Weimar, Heft 4/94, Weimar, 1994

Hauschild, T.; Hübler, R.; Steinmann, F.: „Knowledge Support for functional design of Buildings“, in 6. IABSE Colloquium Reports, Bergamo, 1995

Hübler, R.; Steinmann, F.: „Knowledge-based computer assistance for functionality and shape oriented building design“, in Computing in Civil and Building Engineering“,
Hrsg. J.P.Pahl, Balkema Verlag, Rotterdam, 1995

[1] Fördernummer: HU519/1-4

Steinmann, F.: „Models for live cycle support - Design with building models for a comprehensive support of live cycle phases“,
in "Critical Review of the Applications of Advanced Technologies",
Proceedings of the 5.EuropIA - Lyon, Europia Productions, Paris,1995

Kolbe, P.; Ranglack, D.; Steinmann, F. : „Eine Schnittstelle für dynamische Objektstrukturen für Entwurfsanwendungen“, in Tagungsband 15. IKM, Bauhaus-Uni. Weimar, Feb.1997

Steinmann, F.; Wehner; R.; Hübler, R.: „FLEXOB - Entwicklungstool für dynamische, modellbasierte CAD-Systeme“, in Tagungsband 15. IKM, Bauhaus-Uni. Weimar, Feb.1997

Steinmann, F.; Hübler, R. : „Vorgehensmodelle als Basis der Gestaltung durchgängiger CAD-Systeme“, in Tagungsband 15. IKM, Bauhaus-Uni. Weimar, Feb.1997

Weimar, den 2.5.1997 Unterschrift

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, daß ich die vorliegende Dissertationsschrift zum Thema

Modellbildung und computergestütztes Modellieren
in frühen Phasen des architektonischen Entwurfs

selbständig und nur unter Verwendung der angegebenen Literaturquellen und Hilfsmittel angefertigt habe. Von mir wurde weder diese noch eine andere Dissertation an einer anderen Hochschule, Universität oder wissenschaftlichen Einrichtung zur Einleitung eines Promotionsverfahrens eingereicht.

Weimar, den 4. Mai 1997